

AD-A080 735

SOUTH CAROLINA UNIV COLUMBIA DEPT OF ELECTRICAL AND --ETC F/6 9/2
MULTIPLE MICROCOMPUTER CONTROL ALGORITHM.(U)
SEP 79 R O PETTUS, R D BONNELL, M N HUMMS

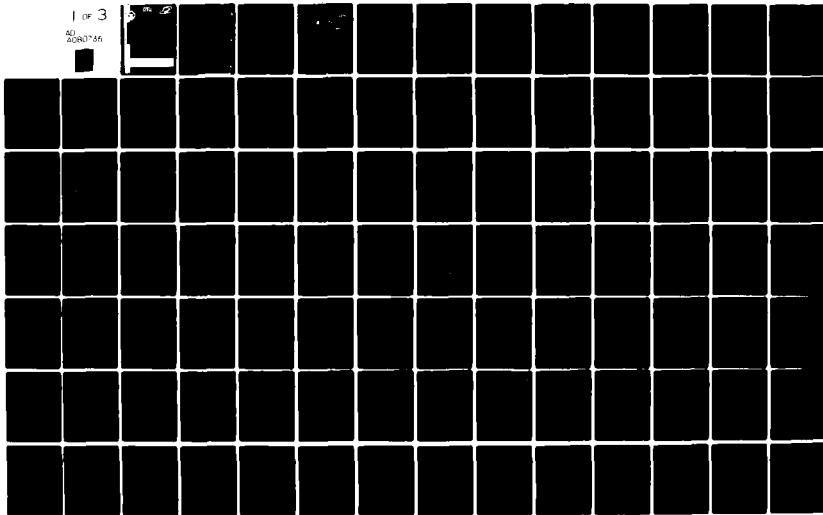
N61338-78-C-0157

UNCLASSIFIED

NAVTRAEQUIPC-78-C-0157-1 NL

1 OF 3

AD-A080 735



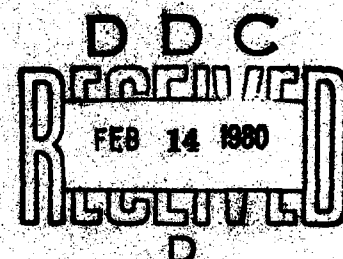
LEVEL II

12

TECHNICAL REPORT NAVTRAEQUIPCEN 78-C-0157-1

MULTIPLE MICROCOMPUTER CONTROL ALGORITHM

R.O. Pettus
R.O. Bonnell
H.N. Huhns
L.M. Stephens
G.M. Wierzba
Department of Electrical and Computer Engineering
University of South Carolina
Columbia, SC 29208



September 1979

Final Report for Period September 1978 through September 1979

DoD Distribution Statement

Approved for Public Release
Distribution is unlimited

THIS DOCUMENT IS BEST QUALITY AVAILABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

ADA080735

80 2 13 027

DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DDC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

UNCLASSIFIED 19

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|-----------------------|---|
| 1. REPORT NUMBER NAVTRAEQUIPCEN 78-C-157-1 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) MULTIPLE MICROCOMPUTER CONTROL ALGORITHM. | | 5. PERFORMING ORG. REPORT NUMBER |
| 6. AUTHOR(s) Robert O. Pettus Ronald D. Bonnell Michael N. Huhns Larry M. Stephens Gregory M. Wierzb | | 7. CONTRACT OR GRANT NUMBER(s) N61338-78-C-157 |
| 8. PERFORMING ORG. ADDRESS University of South Carolina Electrical and Computer Engineering Department Columbia, SC 29208 | | 9. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Experimental Computer Simulation Laboratory Naval Training Equipment Center Orlando, Florida 32813 | | 12. REPORT DATE Sept 79 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | | 13. NUMBER OF PAGES 198 |
| 16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release; Distribution unlimited | | 15. SECURITY CLASS. (of this report) UNCLASSIFIED |
| 17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) | | 18a. DECLASSIFICATION/DOWNGRADING SCHEDULE |
| 18. SUPPLEMENTARY NOTES | | |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Control Structures Partitioning Cache memory Real-time operating systems Microprogramming Virtual machines Multiple computer control algorithms Logical design Microcomputers/microprocessors Real-time systems Computer architecture Control sequences | | |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The design, analysis and performance evaluation of the architecture of a multiprocessor is presented. The architecture is a hierarchically structured, functionally distributed, multiple microcomputer system. Its operating system is a multi-level structure implemented in an optimal combination of hardware, firmware, and software. This architecture is suited to any application, such as process control or real-time system simulation, in which the basic computational tasks do not change in time. | | |

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6001UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

327220

LM

UNCLASSIFIED

Each microcomputer has a dedicated memory space in which program tasks are stored. In addition, there is a system bus to a global memory which is used primarily for communication among the microcomputer. To minimize contention for this system bus, selected areas of global memory are duplicated at each microcomputer. This allows the microcomputers to obtain needed information by using a local bus rather than the global, system bus. All write operations to the shared memory are global and the information is duplicated at microcomputers having that address. Read operations then become primarily local and can occur in parallel.

Control functions are distributed among the microcomputers; however, the scheduling and execution of these tasks is governed at each microcomputer level by a local, real-time operating system. This local operating system is implemented primarily in firmware to minimize overhead. However, the control structure is designed to be independent of implementation so that a variety of microcomputers can be utilized together. Moreover, it is possible to add to each local processor an additional subprocessor which implements the operating system.

| | |
|--|----------------------|
| Accession For | |
| NTIS GRA&I <input checked="" type="checkbox"/> | |
| DDC TAB <input type="checkbox"/> | |
| Unannounced <input type="checkbox"/> | |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist. | Avail and/or special |
| A | 23A |

DDC
RECEIVED
FEB 14 1980
REGULATED

(1)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

NAVTRAEQUIPCEN 78-C-0157-1

TABLE OF CONTENTS

| | | |
|------|--|----|
| I. | INTRODUCTION | 7 |
| | SCOPE | 7 |
| | SYSTEM GOALS | 7 |
| | MULTIPLE MICROCOMPUTER SYSTEM CONFIGURATION. | 7 |
| | PREFERRED ALGORITHM CONCEPT. | 8 |
| | CONCURRENT EXECUTION OF TASKS. | 12 |
| | RESTRICTIONS ON CONCURRENT TASKS | 13 |
| | Shared Resources. | 13 |
| | Process Synchronization | 14 |
| | Semaphores | 14 |
| | Conditional Critical Regions | 14 |
| | Scheduling Mechanisms | 14 |
| | Communications Between Tasks. | 15 |
| | IMPLEMENTATION OF THE NAVTRAEQUIPCEN PREFERRED ALGORITHM | 15 |
| | NOMENCLATURE | 15 |
| II. | TASK PARTITIONING. | 18 |
| | INTRODUCTION | 18 |
| | PARTITIONING | 19 |
| | Acyclic Task Graph. | 19 |
| | Flight Simulation | 22 |
| | Nearly-Decomposable Systems | 25 |
| | Parallelism in a Program Loop | 25 |
| | Optimum Partitioning By Mathematical Optimization | 26 |
| | Exhaustive Scheme for Partitioning. | 28 |
| | PERFORMANCE ANALYSIS OF MULTIPLE MICROCOMPUTER SYSTEM. | 32 |
| III. | CONTROL ALGORITHM IMPLEMENTATION | 32 |
| | FEATURES OF THE CONTROL ALGORITHM. | 32 |
| | Hardware, Software, and Firmware. | 32 |
| | The Virtual Machine Concept | 34 |
| | Shared Memory | 34 |
| | Asynchronous Operation. | 34 |
| | Identical Hardware Modules. | 34 |
| | Hierarchical Organization | 36 |
| | SYSTEM ARCHITECTURE. | 36 |
| | Bus Structure | 36 |
| | Shared Memory Bus Priority Arbitration. | 46 |
| | The Distributed Cache Shared Memory | 49 |
| | The Application Task Manager. | 49 |
| | The Microcomputer Modules | 49 |
| | THE APPLICATIONS TASK MANAGER. | 52 |
| | ATM Functions | 52 |
| | ATM Operation | 52 |
| | Basic Concepts | 52 |

TABLE OF CONTENTS (CONTINUED)

| | |
|---|-----|
| MMCS Interrupts. | 55 |
| The Interrupt Handler and Vector Table | 56 |
| Supervisor Call Handler. | 56 |
| Initialization Routine | 56 |
| The COMMUNICATIONS, HALT, and WAIT States. | 57 |
| Scheduler. | 57 |
| ATM Time-Keeping Function. | 57 |
| Application and System Program Tasks | 58 |
| Tasks States | 58 |
| The ATM Scheduling Mechanism | 58 |
| The Scheduler | 60 |
| The Event Queue Handler. | 60 |
| The System Program Queue | 60 |
| The Application Task Scheduler | 60 |
| ATM Data Structures | 62 |
| The Task Control Block | 62 |
| The Jobs Queue | 65 |
| The Event Queue. | 65 |
| The System-Program Queue | 65 |
| The Ready-Task Queue | 68 |
| ATM Implementation Requirements | 68 |
| ATM Supervisor Calls. | 70 |
| THE CONTROL PROGRAM. | 72 |
| Extension of the Virtual Machine Concept. | 72 |
| The System State/Frame Period. | 72 |
| The Control Processor. | 74 |
| Classification and Scheduling of System Events. | 74 |
| Classification of Events | 74 |
| Scheduling of Events | 76 |
| Control Program Operation and Structure | 77 |
| The Cycle Program. | 77 |
| The Distributed Control Program. | 79 |
| The Exceptional Event Handlers | 79 |
| Diagnostics. | 79 |
| System Initialization. | 79 |
| Control Processor Tasks. | 79 |
| IV. MICROCOMPUTER MODULE DESIGN AND ANALYSIS | 81 |
| ARCHITECTURE OF THE MICROCOMPUTER MODULE | 82 |
| Data Paths to the ALU | 82 |
| ALU Design. | 87 |
| Instruction Decoding Logic. | 87 |
| Microprogram Sequencer and Control Store. | 88 |
| PROCESSOR MODULE FIRMWARE. | 90 |
| Microprogram Structure. | 90 |
| Floating-Point Logic and Microcode. | 98 |
| ATM LOGIC AND MICROCODE. | 103 |
| MICROCODE-CONTROL-STORE MODULE DESIGN. | 103 |
| MEMORY MODULE DESIGN | 103 |

TABLE OF CONTENTS (CONTINUED)

| | |
|--|-----|
| PERFORMANCE EVALUATION | 105 |
| V. CONCLUSIONS. | 113 |
| THE PREFERRED ALGORITHM. | 113 |
| CONTROL ALGORITHM INVESTIGATION. | 113 |
| THE MICROCOMPUTER MODULES. | 114 |
| PARTITIONING THE APPLICATION PROBLEM | 114 |
| SUMMARY. | 115 |
| REFERENCES | 116 |
| APPENDIX A - ATM COMMUNICATION STATE | 118 |
| APPENDIX B - SUPERVISOR CALLS. | 121 |
| APPENDIX C - VAX-EMULATOR DEFINITION FILE. | 123 |
| APPENDIX D - VAX-EMULATOR INSTRUCTION SET. | 141 |
| APPENDIX E - PERFORMANCE ANALYSES | 170 |
| APPENDIX F - APPLICATION TASK MANAGER PASCAL VERSION | 177 |

LIST OF FIGURES

| <u>Figure</u> | | <u>Page</u> |
|---------------|--|-------------|
| 1 | Simplified System Block Diagram. | 8 |
| 2 | Sequential Execution of Tasks. | 10 |
| 3 | Concurrent Execution of Task T_1 to T_N | 10 |
| 4 | An Arbitrary Nesting of Sequential and Concurrent Tasks. | 11 |
| 5 | n-calculations for a Frame Period with Four Cycles. | 17 |
| 6 | Representative Aircraft Type Trainer Simulation Functions. | 20 |
| 7 | Open-loop Data Flow Analysis for Flight Simulation System. | 21 |
| 8 | Flight Simulation System Subsystem Task Graph. | 23 |
| 9 | Acyclic Subsystem Task Graph. | 24 |
| 10 | Virtual Machine State Diagram. | 33 |
| 11 | Role of Virtual Machine in Transfer of Task Variables. | 35 |
| 12 | Hierarchical Organization of MMCS. | 37 |
| 13 | MMCS Block Diagram. | 38 |
| 14 | Signal Structure of the Bus Arbitration Lines. | 39 |
| 15 | Operation of Shared Memory Priority Scheme. | 41 |
| 16 | Flowchart of BAM Operation. | 42 |
| 17 | BAM State Diagram. | 43 |
| 18 | Simplified Block Diagram of BAM Logic. | 44 |
| 19 | Flowchart for Operation of Processor Module Priority Logic. | 45 |
| 20 | Illustration of Distributed Cache Allocation. | 47 |
| 21 | Role of ATM in MMCS. | 50 |
| 22 | Processor Module and Bus Interface. | 51 |
| 23 | Simplified Flowchart of ATM Operation. | 53 |
| 24 | Complete ATM Flowchart. | 54 |
| 25 | Task State Diagram. | 59 |
| 26 | Flowchart of Scheduler Portion. | 61 |
| 27 | Task Control Block. | 63 |
| 28 | Task Control Block State Flags. | 64 |
| 29 | Event Queue Structure. | 66 |
| 30 | Structure of The Systems Program Queue. | 67 |
| 31 | Ready Task Queue Structure. | 69 |
| 32 | SVC Opcode Byte. | 71 |
| 33 | Extension of the Virtual Machine Concept to the Control Algorithm. | 73 |
| 34 | Placement of Event Classes Within MMCS Structure. | 75 |
| 35 | Organization of the Cycle Program. | 78 |
| 36 | Hardware Block Diagram of a Processor Module. | 83 |
| 37 | Data Paths to the ALU. | 84 |
| 38 | Block Diagram of Arithmetic-Logic Unit. | 85 |
| 39 | Block Diagram of Instruction Decoding Logic. | 86 |
| 40 | Microprogram Sequencer and Control Store. | 89 |
| 41 | Functional Block Diagram of the Microprogram Structure. | 93 |
| 42 | Flowchart for the Main Microprogram. | 94 |

LIST OF FIGURES (CONTINUED)

| | | |
|-----|--|-----|
| 43 | Flowchart for the Microsubroutine for the MOV Instruction . . | 95 |
| 44 | Microsubroutine for the Register-direct Source Operand | 96 |
| 45 | Microsubroutine for Register-direct-mode Destinations. | 97 |
| 46 | Microword Format for Bits 111 - 72 | 99 |
| 47 | Microword Format for Bits 71 - 32. | 100 |
| 48 | Microword Format for Bits 31 - 0 | 101 |
| 49 | Microcode Control Store PROM Module. | 104 |
| 50 | Microcomputer Module Block Diagram and Bus Interfaces. | 106 |
| 51 | Input and Output Signals for the Memory-Alignment Unit | 107 |
| 52 | ASM Chart for Memory-Alignment Unit Controller | 108 |
| 53 | Data Rearrangement During Both Read and Write Operations Using the Memory-Alignment Unit. | 109 |
| 54 | Hardware Logic for RAM Write Enables | 110 |
| A-1 | Assignment of control bus memory space to the application processors | 120 |

NAVTRAEQUIPCEN 78-C-0157-1

LIST OF TABLES

| <u>Table</u> | | <u>Page</u> |
|--------------|--|-------------|
| 1 | All Possible Partitions, B ₃ ⁴ - 14. | 27 |
| 2 | Bus Request Assignments | 48 |
| 3 | VAX 11/780 Instructions Microcoded. | 91 |
| 4 | Instruction Times and Usage Factors | 112 |
| E-1 | Details of AIET Calculations. | 172 |
| E-2 | Selected Addressing Modes Execution Steps | 176 |

SECTION I

INTRODUCTION

SCOPE

The purpose of this investigation was to derive a control algorithm for a new concept in computer architecture developed at the Naval Training Equipment Center (NAVTRAEQUIPCEN), Orlando, Florida. The proposed architecture is a hierarchically structured, functionally distributed, multiple microcomputer system and will be referred to as a multiple microcomputer system (MMCS). The MMCS control algorithm is a multilevel structure implemented by an optimal combination of hardware, firmware (microcode), and software. This report documents the results of the total investigation and the recommended design approach.

SYSTEM GOALS

The goals of the multiple microcomputer control algorithm are to:

- a) Reduce programming and program maintenance costs of the software for real-time trainers.
- b) Offer increased standardization and modularity.
- c) Improve system throughput.
- d) Provide a basis for future system improvement.

MULTIPLE MICROCOMPUTER SYSTEM CONFIGURATION

The MMCS configuration is shown in Figure 1. The major system hardware components are N microcomputer modules, a shared memory, a bus arbitration module, and a system communication and control bus. The microcomputer modules (also referred to as processor modules) are implemented with a high-performance microprocessor technology. The number of microcomputer modules required is a function of the amount of computation required by a specific trainer system. A portion of the MMCS architecture control algorithm is implemented by the executive software for each processor module. This executive, designated the applications task manager (ATM), consists of both native code and microcode. Two key features of the architecture are the existence of a control processor which is capable of directing the activities of the remaining processors, and the distribution of processing tasks by function. The details of the MMCS configuration are discussed in Sections III and IV of this report.

PREFERRED ALGORITHM CONCEPT

The technical approach suggested by NAVTRAEQUIPCEN has been evaluated and found to be a desirable architectural concept. Implementation of the preferred algorithm depends upon two major criteria:

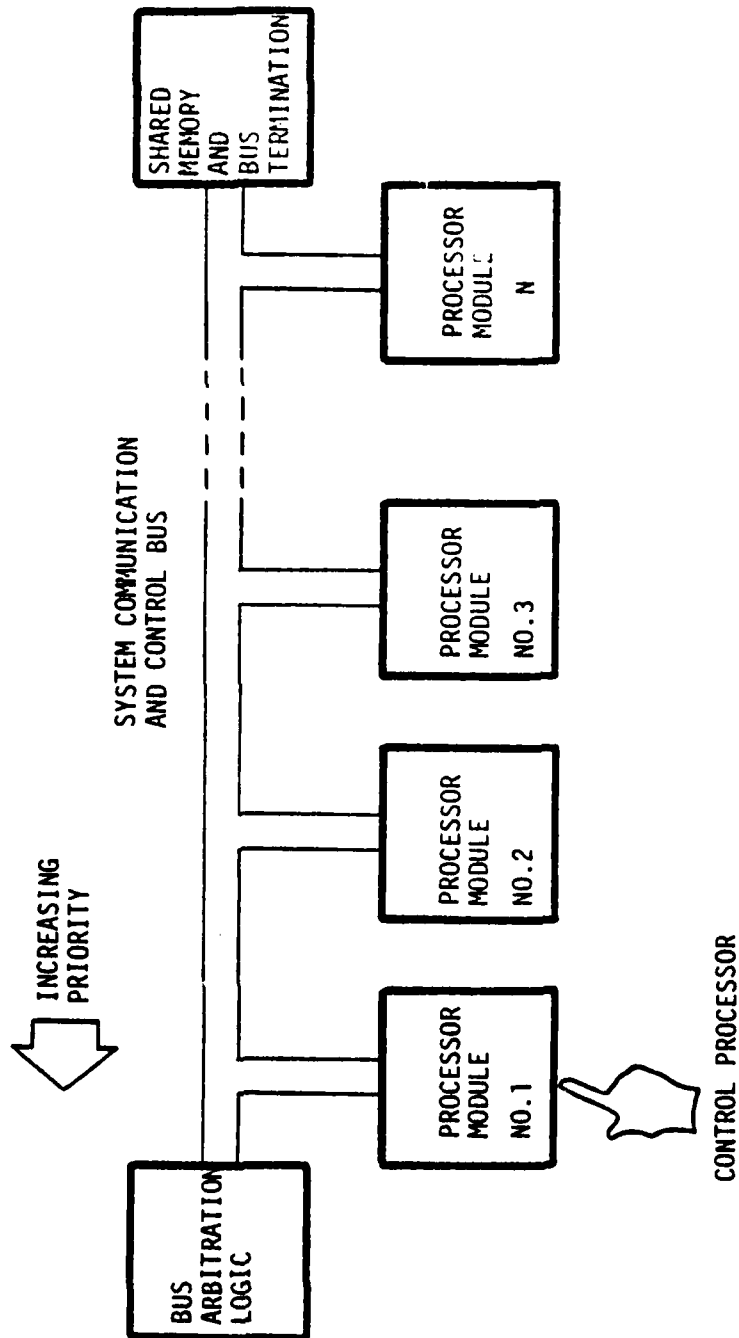


Figure 1. Simplified System Block Diagram

- a) Successful partitioning of the problem into disjoint tasks.
- b) Developing a run-time structure which provides for the passing of system parameters among processors while preserving precedence.

Both of these criteria must be met without any serious impact upon the application programmers. Criterion (a) is discussed in Section II; criterion (b) is the major topic of this report and the general concepts used to satisfy it are presented in this section. This involves:

- a) the basic conditions required for the concurrent execution of system tasks.
- b) a technique for handling shared resources, primarily shared data.
- c) synchronizing techniques for maintaining system precedence.

CONCURRENT EXECUTION OF TASKS

The notation

BEGIN T1; T2;.....; Tn; END (1)

indicates that the tasks T1, T2,....., TN are executed sequentially in the order given. The notation

BEGIN T0; COBEGIN T1; T2;.....; Tn; COEND Tn+1; END (2)

indicates that tasks T1 through Tn may be executed concurrently. (A task is defined to be the smallest system entity capable of contending for system resources). Precedence graphs of statements 1 and 2 are given in Figures 2 and 3. Concurrent and sequential statements may be arbitrarily nested, as given in statement 3 below and shown in Figure 4.

```

COBEGIN
  T1;
  BEGIN
    T2;
    COBEGIN T3; T4; COEND
    T5;
  END
  T6;
COEND
    
```

(3)

Concurrent tasks may be executed by a multiple processor system or by time-slicing on a single processor. In terms of the basic definition there is no difference. In practical systems multiple processors offer the possibility of improved performance while time-slicing in a single processor improves its utilization.



Figure 2. Sequential Execution of Tasks

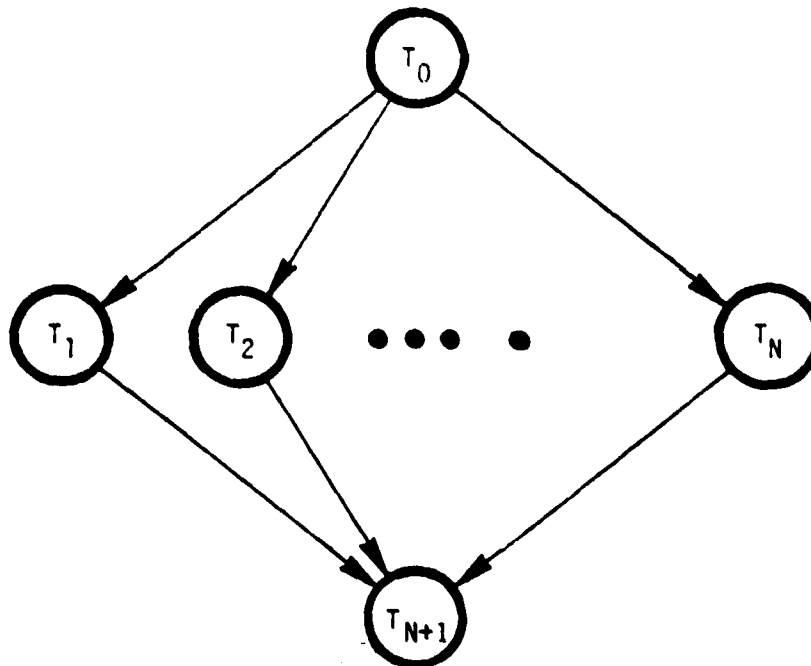


Figure 3. Concurrent Execution of Task T_1 to T_N

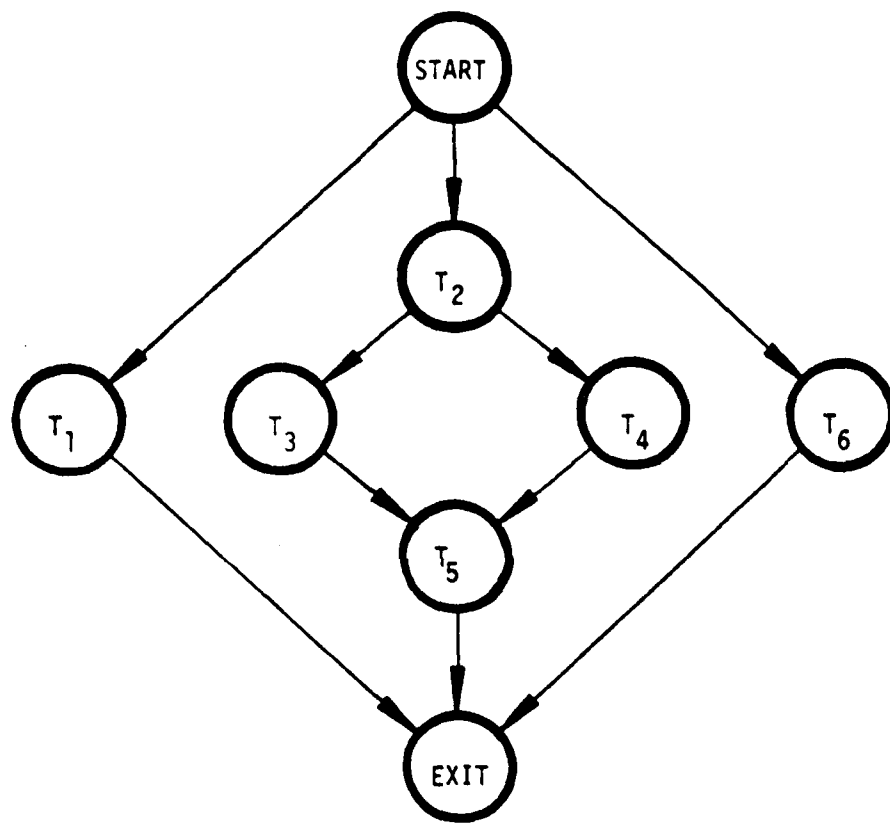


Figure 4. An Arbitrary Nesting of Sequential and Concurrent Tasks

RESTRICTIONS ON CONCURRENT TASKS

Certain restrictions must be placed upon systems of concurrent tasks (or programs) to insure that reasonable performance verification and program testing techniques may be used. The first restriction is that a transfer of control from a concurrent task is prohibited. This restriction is required because:

- a) A concurrent statement (COBEGIN...COEND) is terminated only when all of its tasks are terminated.
- b) A statement having one or more concurrent sections embedded may continue as a sequential process after one or more of the current sections is terminated.

This restriction does not impose any problems in the use of systems of concurrent tasks. In fact the same restriction is typically imposed by the use of good programming practices.

A second restriction concerns the independence of the tasks enclosed in a concurrent statement. It is desirable for a system of concurrent tasks to be testable as a unit. Consider a group of concurrent tasks $T_1; \dots; T_n$; where each T_i has a predicate P_i and a result R_i such that if P_i is true before execution of T_i , then R_i will be true afterwards. Then the desired property is that the concurrent task

COBEGIN $T_1; T_2; \dots; T_n$; COEND

be considered as a single operation S with predicate P and result R where

$P = P_1 \text{ and } P_2 \text{ and } P_3 \dots \text{ and } P_n$
 and $R = R_1 \text{ and } R_2 \text{ and } R_3 \dots \text{ and } R_n.$

A second desirable characteristic for a system of concurrent tasks is that it be functional. A functional system is one where the output history of its execution is a time-independent function of its input history. Functional behavior, together with the ability to apply a single predicate and result to a system of concurrent tasks, is required for the application of rigorous techniques of program verification and testing. It may be shown that a sufficient condition for these characteristics to hold is that the tasks be disjoint.¹ This means that a variable v_i changed by task T_i cannot be referenced or changed

1. Hansen, Per Brinch, Operating System Principles, N.J., Prentice-Hall, 1973, pp 60-76.

by another task T_j ($i \neq j$). This does not preclude concurrent tasks from referencing common variables not changed by any of them.

However, in dealing with real-time trainer system programs (and, in fact, most other real-time applications) the disjointness restriction is difficult to meet. In systems of this type it is vital to pass parameters between tasks. In addition, trainer systems use a highly iterative system, the sampling rate being a function of the natural frequency of the system being simulated. Under these conditions the tasks may not be functional during a given sampling interval.

It will be necessary for the control algorithm to provide mechanisms by which the restriction of disjointness may be bypassed without giving up the characteristics of functional behavior or a combined predicate-result. There are two separate problems; providing for shared resources, i.e., variables common to more than one task, and maintaining the long-term precedence of the system. The term long-term precedence concerns the system behavior over periods of time which are longer than the sampling period. These mechanisms are discussed in the following sections.

Shared Resources

If more than one task is to share a resource (typically a variable) then all but one task must be excluded from operating on the resource at any one time. The technique for doing this is the critical region. The notation

REGION v do T_i

associates a task T_i with a variable v . Critical regions referring to the same variable (or resource) exclude one another in time. In particular the following conditions apply to critical regions:

- a) Only one task may be within a critical region at a time.
- b) When a task wishes to enter a critical region, it will be enabled to do so in a finite length of time, i.e., the scheduling algorithm must be fair.
- c) A process may remain in a critical region for a finite length of time only. This implies that all tasks operating on a common variable must terminate.

It should be noted that while critical regions exclude each other in time, they imply nothing about the sequencing of tasks.

Process Synchronization

When a single task is partitioned into a group of concurrent tasks we need a synchronizing technique to preserve the precedence of the system. Two schemes for doing this are the semaphore and the conditional critical region.

Semaphores. The semaphore is a mechanism whereby a timing signal may be sent from one process to another. If the variable v is a semaphore, then the two permissible operations are $SIGNAL(v)$ and $WAIT(v)$. The semaphore v is characterized by the integers:

$S(v) :=$ number of signals sent
 $R(v) :=$ number of signals received
 $C(v) :=$ number of initial signals

where the initial assignment is given by $v=c$. The relationship of statement

$$0 \leq R(v) \leq S(v) + C(v) \leq R(v) + MAX$$

must always hold for the semaphore v . MAX is the maximum value v is allowed to have. This relationship is the semaphore invariant. Since the semaphore is a common variable for the senders and receivers, the signal and wait operations exclude each other in time.

Conditional Critical Regions. The conditional critical region is a mechanism for synchronizing a task's entry into a critical region. It is denoted $AWAIT$ and used as follows:

REGION v DO
 BEGIN $AWAIT$ (CONDITION);...END

The condition must be a Boolean expression or have a Boolean result. The task does not enter the critical region until the condition is true.

Scheduling Mechanisms

Several of the functions previously defined are required to control the scheduling of shared resources. The basic scheduling mechanism of the preferred control algorithm is the queue. The permitted operations on a queue, q , are $ENTER(t,q)$ and $REMOVE(t,q)$ which enter and remove an element t from q according to the desired scheduling policy. In addition the Boolean variable $EMPTY(q)$ denotes whether the queue q is empty or not.

Communications Between Tasks

While the rates at which the tasks of a trainer system proceed are related over the long term, they may be independent over the short term (i.e., times on the order of the sampling period). This complicates the handling of inter-task communication. When a sending task produces a message, the receiving task may not be ready for it. Since delaying the sender would decrease throughput, message buffers are used for intertask communications. A message buffer is

NAVTRAEQUIPCEN 78-C-0157-1

designated B and has capacity MAX. The operations SEND(M,B) and RECEIVE(M,B) place the message M on the buffer or remove it, respectively. These operations, like SIGNAL and WAIT, must exclude each other in time and are critical regions with respect to the buffer used.

IMPLEMENTATION OF THE NAVTRAEQUIPCEN PREFERRED ALGORITHM

The control algorithm is implemented as a mix of hardware, firmware, and software. In particular:

- a) The critical region concept is essential and basic. It will be implemented in hardware such that it is an inherent characteristic of the system.
- b) The synchronizing tools are also essential but are much more complex. They will be implemented in firmware (microcode).
- c) The scheduling policy may change from problem to problem. This will be implemented in software and be accessible to the user.

Details of the implementation of these techniques are given in the remainder of the report.

NOMENCLATURE

In the following, several terms are defined which are useful in describing the timing of the proposed multicomputer system:

Task = a self-contained portion of a computation which once initiated can be carried out to its completion without the need for additional inputs.

Frame period, T = the physical sampling period. It is a function of the significant highest natural frequency, ω_n , of the system being simulated. In the trainer simulation problem, twenty times the highest natural frequency is used. The frame period is equal to:

$$T = \frac{1}{20 \omega_n / 2\pi} \quad (5)$$

where $20 \frac{\omega_n}{2\pi} \geq 30 i$, i = positive integer.

The (30 i) constraint is due to consideration of any visual system employing the standard TV raster frame rate.

Cycle period, c = the time allowed to complete a write cycle plus a read cycle on the common control bus. During each cycle a set of calculations are completed by the multiple processors. The cycle period is equal to

$$c = \frac{T}{N_c} \quad (6)$$

where N_c = the number of cycles within a frame period.

- n-calculations = those calculations which must be completed once every frame period.
- k-calculations = those calculations which must be completed once every k frame periods.
- Synchronous n-calculations on a cycle period = those n-calculations which are completed synchronously on a cycle period but are asynchronous within the cycle period.
- Update state n-calculations - those zero-memory n-calculations which must be completed before the next states can be calculated. These calculations can be done in either the individual application processors or the input processor.
- Next state n-calculations = those memory n-calculations which must be completed to determine the next state.
- Output n-calculations = those zero memory n-calculations which must be completed to determine the outputs or some input variables for the next frame. These calculations can be done in either the individual application processors or the output processor.

A typical n-calculation for a frame period with four cycles is shown in Figure 5. The abbreviations used in Figure 5 are defined as follows:

- AP = applications processor
- SM = shared memory
- OP = output processor
- IP = input processor

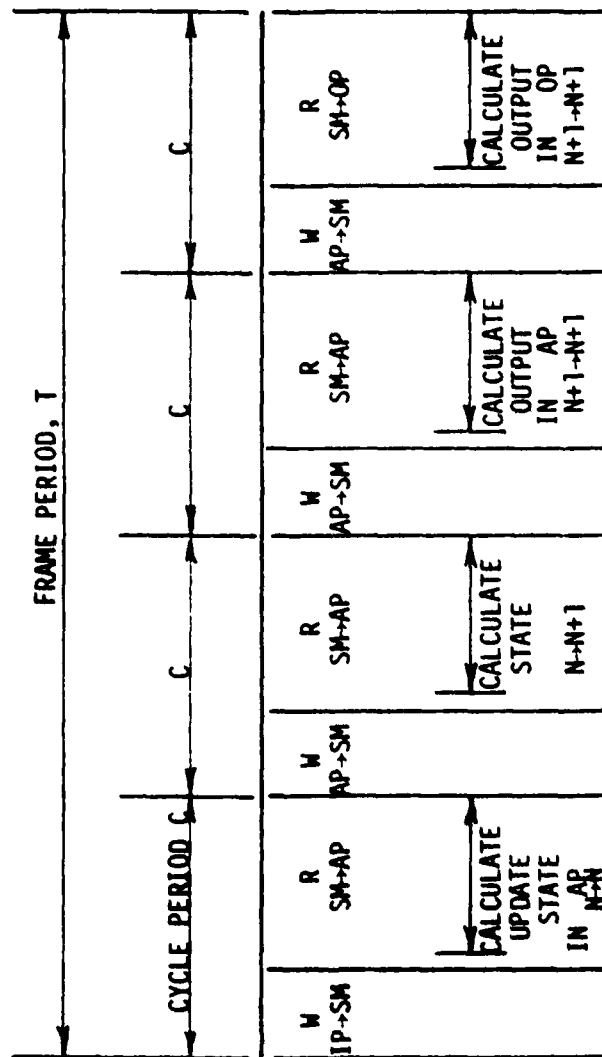


Figure 5. n-calculations for a Frame Period with Four Cycles.

SECTION II

TASK PARTITIONING

INTRODUCTION

Parallel processing is more complex than sequential processing because of the need for communications. The analysis of parallel algorithms hence must include the effect of communications between tasks. The number of data transfers on the bus is a measure of the communications inherent in the algorithm because as this number increases the necessary communications increase. In general, these communications should be minimized while at the same time keeping the computation time as short as possible.² The scheduling and allocation of tasks to processors in such a way that some objective (e.g., minimum execution time or minimum communication time) is realized is an important area of parallel processing and one fundamental to developing a control algorithm for a functionally distributed system of the type examined in this contract. As per Section 3.5 of Specification N-74-105,³ the data employed for problem partitioning are typical of the simulation problem requirements of a modern high-performance trainer. Appendices C and D of Specification N-74-105 provide the data used in this study.

PARTITIONING

Two tasks are parallel if and only if they produce the same result when performed sequentially or in parallel for all possible sets of input data. In general, the parallelism of two tasks can be shown to be undecidable.⁴ The question of whether or not two tasks can be performed in parallel is a function of not only the algorithm but also of the way it has been programmed. It should be noted that the partitioning of a trainer simulation problem is for a fixed and known program.⁴ Most of the partitioning algorithms have been developed for the partitioning of a general unspecified program. In order to achieve the best partition, use should be made of all knowledge about the program.

2. Agerwala, T. and Lint, B., "Communication in Parallel Computer Systems", Proc. 1978 Conf. Information Science and Systems, Johns Hopkins University, March 29-31, 1978, pp. 89-95.

3. Sumner, C.F., Specification for Multiple Microcomputer Control Algorithm Investigation, NTEC N-74-105, May 22, 1978.

4. Bernstein, A.J., "Analysis of Programs for Parallel Processing", IEEE Trans. on Electronic Computers, Vol. EC-15, No. 5, Oct. 1966, pp. 757-763.

Acyclic Task Graph

A program task graph is a graph structural model of a program exhibiting the flow relation or connection among the tasks in the program. Computational processes can be modelled by directed graphs in which the nodes represent single tasks at a specified structural level and the directed branches represent the permissible transition to the next task in sequence. A graph consisting of a set of nodes is said to be strongly connected if and only if any node in it is reachable from any other.

A program task graph which has maximal strongly connected subgraphs can be reduced to an acyclic task graph by replacing each of the maximal strongly connected subgraphs by a single node. By using the reduced acyclic task graph it can be shown that the problem of parallelism becomes decidable.⁵

For programs which can be represented by acyclic task graphs, algorithms have been designed to determine the minimum number of processors needed to execute the program in the shortest time. Other algorithms that have been developed for the acyclic task graph are (1) a determination of the minimum time to process a graph, given k processors; (2) a determination of whether or not a graph can be processed in the minimum possible time with k processors; and (3) a determination of lower and upper bounds on the minimum number of processors required to process an acyclic task graph in the shortest possible time.⁶

Flight Simulation

A typical aircraft type trainer simulation, as shown in Figure 6, may consist of as many as four related major system simulations. The coupling between these real-time functions is highly synchronized and very tight. All four major functions depicted in Figure 6 are not present in all aircraft type trainers. In Addendum D of Specification N-74-105 a flight simulation flow diagram is specified. From an open-loop input-output viewpoint, it has two inputs and four outputs, as shown in Figure 7. This particular flight simulation system has seven identifiable subsystems:

a) Equations of motion

5. Ramamoorthy, C.V., and Gonzalez, M.J., "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs", 1969 Fall Joint Computer Conference, AFIPS Conf. Proc., Vol. 35, Montvale, NJ: AFIPS Press, 1969, pp. 1-15.

6. Ramamoorthy, C.V., Chandy, K.M., and Gonzalez, M.J., "Optimal Scheduling Strategies in a Multiprocessor System", IEEE Trans. on Computers, Vol. C-21, No. 2, Feb., 1972, pp. 137-146.

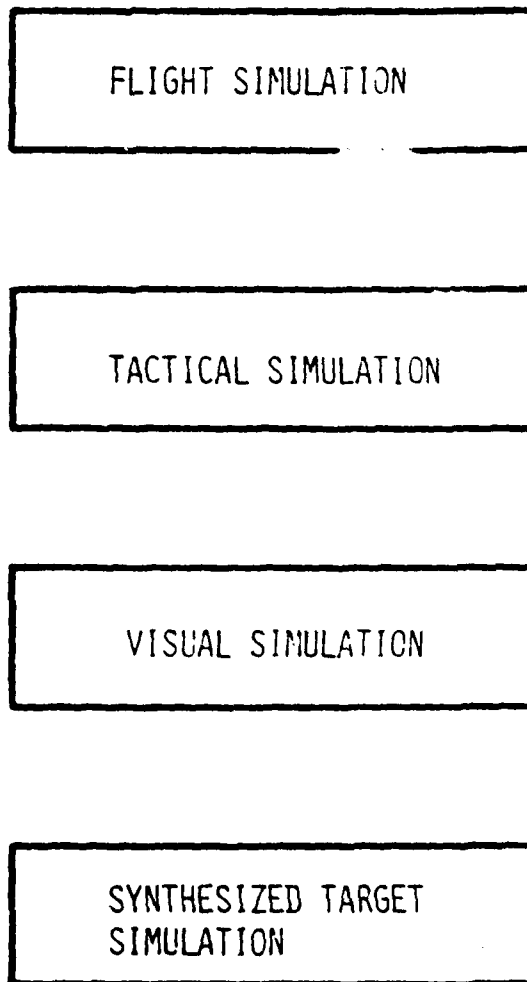
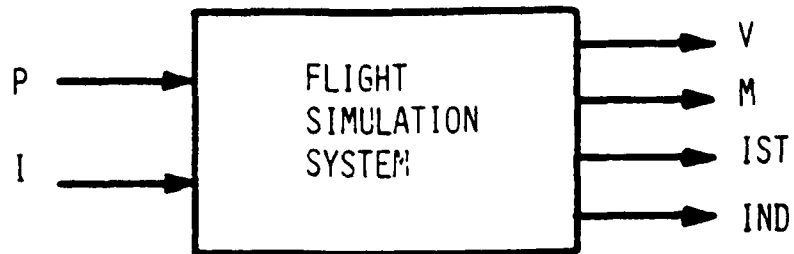


Figure 6. Representative Aircraft Type Trainer
Simulation Functions



P - PILOT
I - INSTRUCTOR
V - VISUAL SYSTEM
M - MOTION SYSTEM
IST - INSTRUMENTS
IND - INDICATORS

Figure 7. Open-loop Data Flow Analysis for Flight Simulation System

- b) Aerodynamics
- c) Propulsion
- d) Weight and balance
- e) Landing gear
- f) Automatic flight control
- g) Control loading.

The subsystem task graph for this flight simulation system is shown in Figure 8. Each node represents a subsystem of the flight simulation system and each directed arc represents data flow from the antecedent node to the consequent node. It is noted that the subsystem task graph is highly coupled and has a total of ten loops

a-b-a
a-c-a
a-e-a
a-f-a
a-c-d-a
a-c-d-e-a
a-f-b-a
a-f-c-a
a-f-c-d-a
a-f-c-d-e-a

If this subsystem task graph is converted to an acyclic subsystem task graph, then only two nodes remain, Figure 9. This result indicates that no parallelism exists at the subsystem structural level for this flight simulation system.

The flight simulation flow diagram has seventy-six distinct tasks at the block diagram structural level. The block task graph has a very large connectivity with many loops. Thus, the acyclic block task graph approach is not applicable for partitioning for parallelism at the block diagram structural level for this flight simulation system.

Nearly-Decomposable Systems

A software product can be considered as a complex dynamic system that evolves with time, where complexity is defined as the number of tasks in the system, and the connectance and strength of the interactions between these tasks. If the complexity of a software system increases to an unmanageable point, then it has in some sense reached a point of instability and the system is incomprehensible. One of the major objectives is to reduce the complexity of the total system software. This can be achieved with a multiple microcomputer

7. Baker, K., "Improve Complex Software by Using Multiple Microprocessors", Microprocessors, Vol. 1, No. 3, Feb., 1977, pp. 165-168.

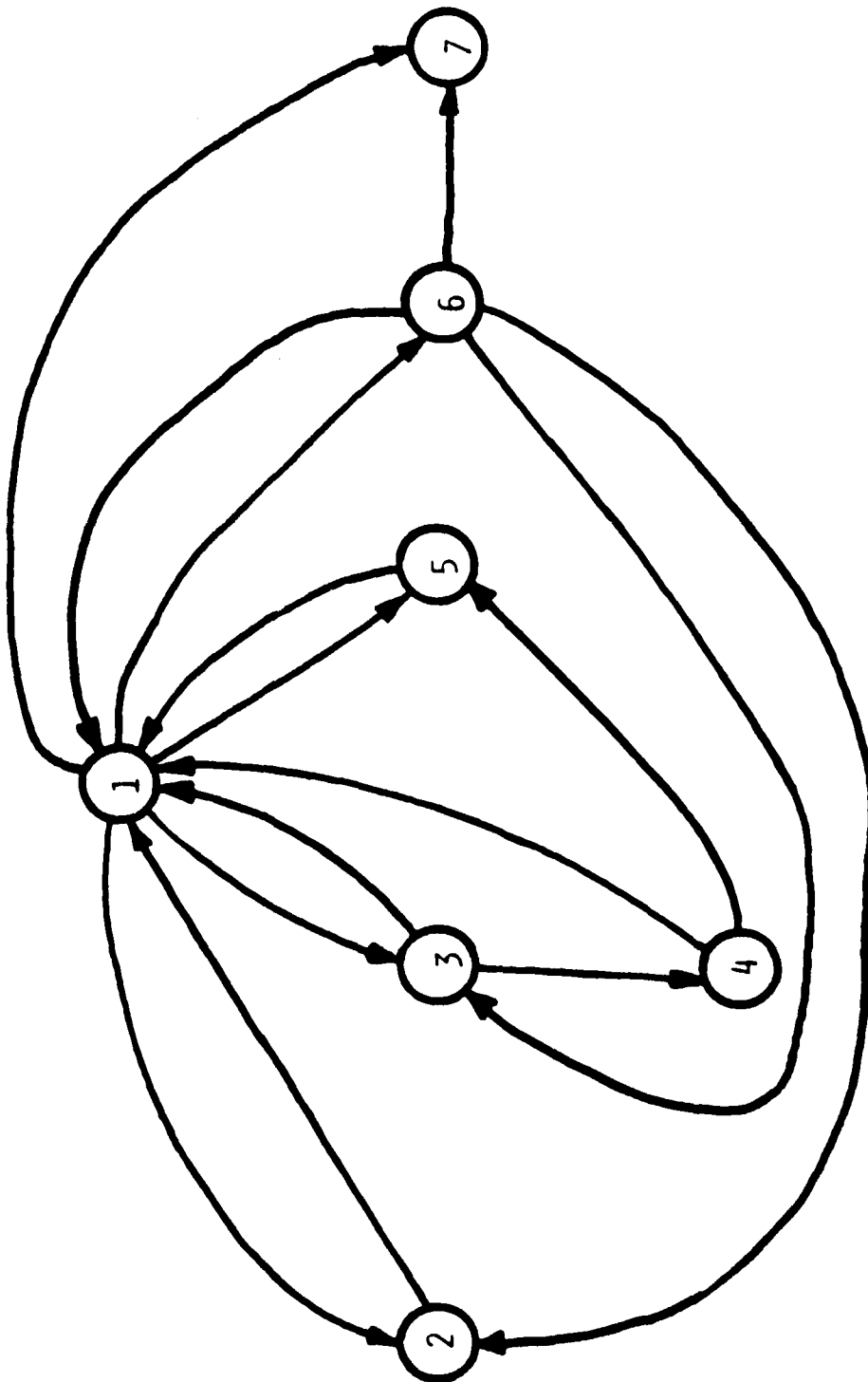


Figure 8. Flight Simulation System Subsystem Task Graph

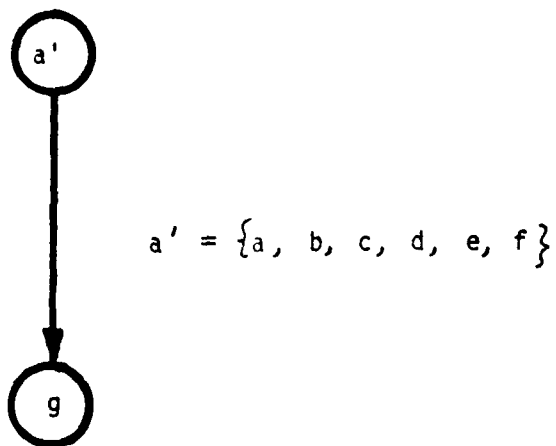


Figure 9. Acyclic Subsystem Task Graph

system because the various tasks (functions) can be distributed over several processors.

A system is said to be "nearly decomposable" if subsystems can be identified where the interactions between subsystems are one or more orders of magnitude less than the interactions within subsystems.⁸ If the original software system is "nearly decomposable", then a multiple microcomputer system architecture results in a decrease in the overall software system complexity. Hence a system with a faster execution time and less complex software can be realized.

Clustering is an approach which may result in a good partitioning of a "nearly-decomposable" system at the program task level. An examination is in progress of this pattern recognition technique but at present a suitable feature space defined on the local properties of the individual tasks has not been found. One heuristic approach which seems plausible is that those tasks that make up a given physical subsystem or its simulation would have more interaction at the task level than the interactions between the different physical subsystems. This, of course, assumes that a task still retains some physical meaning and can be related to the physics of the simulation.

Parallelism in a Program Loop

Some work has been completed where parallel execution was carried down to the level of individual operations in assignment statements of the source program. This is necessary if one wants to execute programs in a faster way. Also, approaches for introducing more parallelism into a program loop (with or without conditional statements such as IF's and GOTO's) have been examined. A loop distribution algorithm has been developed for a program task graph, but to achieve higher speedup with this algorithm, a multiple-array processor is required.

Optimum Partitioning By Mathematical Optimization

Another approach for partitioning a large software system over multiple computer systems is that of mathematical programming. The integer-programming partitioning approach is presently being researched by the U.S. Air Force¹⁰.

8. Simon, H.A., The Science of the Artificial, Cambridge, MA: MIT Press, 1969.

9. Kuck, D.J., "Parallel Processing of Ordinary Programs", Advances in Computers (M. Rubinoff and M.C. Yovits, ed.'s), Vol. 15, N.Y.: Academic Press, 1976, pp. 119-179.

10. Cylmer, S.J., and Price, P.E., "Partitioning Software for Advanced Simulation Computer Systems", to be published, 1979.

where an index to rank performance is designed to include such effects as:

- a) task execution time
- b) task memory utilization
- c) bus utilization time
- d) balance of the load among the processors.

The selection of the "best" partition is subject to two constraints:

- a) real-time task resource requirements
- b) predicted performance simulation feedback.

The results of this research will be available by late Fall, 1979.

Exhaustive Scheme for Partitioning

A study has been made to determine if an exhaustive scheme could be implemented where all possible partitions are considered and the "best" partition based on a performance index is selected. The number of ways B_p^N , of distributing N distinguishable tasks into p indistinguishable processors with empty processors allowed is given by¹¹.

$$B_p^N = \sum_{j=1}^p S(N, j) \quad \forall N \geq p$$

where

$$S(N, j) = \frac{1}{j!} \sum_{i=0}^j (-1)^i \binom{j}{i} (j-i)^N$$

and is called the Stirling number of the second kind.

For example, the number of ways of distributing $N = 4$ distinguishable tasks (T_1, T_2, T_3, T_4) into $p = 3$ indistinguishable processors (P_1, P_2, P_3) is $B_3^4 = 14$. All possible partitions for this example are shown in Table 1.

Next a computer program was developed for an algorithm given by Wilf¹² to determine all Stirling numbers and to calculate B_p^N . As an example, a system

11. Liu, C.L., Introduction to Combinatorial Mathematics, N.Y.; McGraw-Hill, 1968, pp. 38-40.

12. Wilf, H.S., and Nijenhuis, A., Combinatorial Mathematics, N.Y.; Academic Press, 1975, pp. 110-117.

TABLE 1. ALL POSSIBLE PARTITIONS, $B_3^4 = 14$

| P_1 | P_2 | P_3 |
|-------|------------|----------------------|
| 0 | 0 | T_1, T_2, T_3, T_4 |
| 0 | T_1 | T_2, T_3, T_4 |
| 0 | T_2 | T_1, T_3, T_4 |
| 0 | T_3 | T_1, T_2, T_4 |
| 0 | T_4 | T_1, T_2, T_3 |
| 0 | T_1, T_2 | T_3, T_4 |
| 0 | T_1, T_3 | T_2, T_4 |
| 0 | T_1, T_4 | T_2, T_3 |
| T_1 | T_2 | T_3, T_4 |
| T_1 | T_3 | T_2, T_4 |
| T_1 | T_4 | T_2, T_3 |
| T_2 | T_3 | T_1, T_4 |
| T_2 | T_4 | T_1, T_3 |
| T_3 | T_4 | T_1, T_2 |

with five processors ($p = 5$) was chosen and the number of tasks was varied from 10 to 50 and B_p^N calculated. The results are:

| Task | No. of Partitions, B_5^N |
|------|----------------------------|
| 10 | 1.8×10^4 |
| 20 | 1.59×10^{11} |
| 30 | 1.55×10^{18} |
| 40 | 1.52×10^{25} |
| 50 | Computer Overflow Error |

From this study it is concluded that an exhaustive scheme is not feasible for partitioning when the number of tasks is much greater than 15 and the number of processors is between 5 and 10. The flight simulation program with its seventy-six tasks far exceeds these limits so some nonexhaustive scheme must be developed for its partitioning.

PERFORMANCE ANALYSIS OF MULTIPLE MICROCOMPUTER SYSTEM

As has been noted by Kober,¹³ how well the processing power of a multiple computer system can be utilized (i.e., its efficiency) is a function of three major factors:

- The organization and architecture of the system.
- The number and power of the individual processors.
- The type of application program.

One measure of the efficiency of a multiple computer system is the speed-up factor, B , defined as:

$$B = \frac{T_S}{T_P}$$

13. Kober, R., "A Fast Communication Processor for the SMS Multimicro-processor System", Second Symposium on Micro-Architecture, North Holland Publ. Co., 1976, pp. 183-189.

where T_S = the execution time needed for the sequential computation of the application program.

T_P = the execution time needed for the parallel computation of the application program.

In this section, the speed-up factor and the quantities affecting it are examined.

For the multiple microcomputer system architecture presented in this report, a cycle is the time allowed to complete a write (W), a read (R) on the global shared-memory bus, plus a set of state calculations by the individual processors. Because the total computation is performed by a repetitive sequence of cycles, the speed-up factor is based on only one cycle.

Consider a multiple microcomputer system which has n individual processors and a total computational load of M tasks. The average computation time for one task is denoted by T_A . The average time for data exchange on the shared-memory bus per task with only global shared memory is denoted by T_C . The average time for data exchange on the shared-memory bus per task with both local and global shared memory, T_C' , is given by

$$T_C' = k T_C$$

where k is the local shared memory factor ($0 < k \leq 1$). A lower (but not the least lower) bound for k is $1/n-1$. Note, k is a function of the system partitioning.

If $k=1$, there is no local shared memory and shared variables are communicated only through a global shared memory. For $k < 1$ the average time for data exchange on the shared-memory bus is reduced by the presence of the local shared memory.

The average processor utilization for computation, α , is given by

$$\alpha = \frac{T_A}{T_M} \quad (0 < \alpha \leq 1)$$

where T_M = the maximum time allowed for computation. Given the above parameters T_A , k , T_C , n , M , and α , the speed-up factor for the multiple microcomputer system with distributed control, S_d , can be determined.

The execution time needed for sequential computation of M tasks is given by

$$T_S = M T_A$$

The parallel computation time for M tasks by a multiple microcomputer system with distributed control is

$$T_{pd} = M T_C' + \frac{M}{na}$$

Therefore, the speed-up factor S_d is given by

$$S_d = \frac{T_S}{T_{pd}} = \frac{1}{\frac{1}{na} + \frac{T_C'}{T_A}} \quad (\text{lower bound})$$

The speed-up factor for the multiple microcomputer system without distributed control, S_d , is given by

$$S_d = \frac{M T_A}{T_D + M T_C' + \frac{M}{na} T_A}$$

where T_D = duration of control phase.

The speed-up factor is improved by the amount γ , with the use of distributed control

$$\gamma = \frac{S_d}{S_d'} = 1 + \frac{T_D}{M T_C' + \frac{M}{na} T_A}$$

For the proposed single bus multiple microcomputer system and a specified partition, the ratio of the average throughput with bus contentions to the average throughput without contentions, has also been determined^{14,15}.

14 Fung, K.T., and Torng, H.C., "On the Analysis of Memory Conflicts and Bus Contentions in a Multi-Microprocessor System", IEEE Trans. on Computers, Vol. C-27, No.1, Jan., 1979, pp. 28-37.

15 Reyling, G., "Performance and Control of Multiple Microprocessor Systems", Computer Design, March, 1974, pp. 81-86.

The speed-up factor for the multiple microcomputer system with distributed control and with local shared memory, s_d^w , is given by

$$s_d^w = \frac{1}{\frac{1}{na} + \frac{kT_C}{T_A}} \quad 0 \leq k \leq 1$$

The speed up factor without local shared memory, \bar{s}_d^w is given by

$$\bar{s}_d^w = \frac{1}{\frac{1}{na} + \frac{T_C}{T_A}}$$

Therefore the speed-up factor is improved by the amount σ through the use of local shared memory and distributed control

$$\sigma = \frac{1 + C}{1 + kC}$$

where

$$C = na \frac{T_C}{T_A}$$

SECTION III

CONTROL ALGORITHM IMPLEMENTATION

FEATURES OF THE CONTROL ALGORITHM

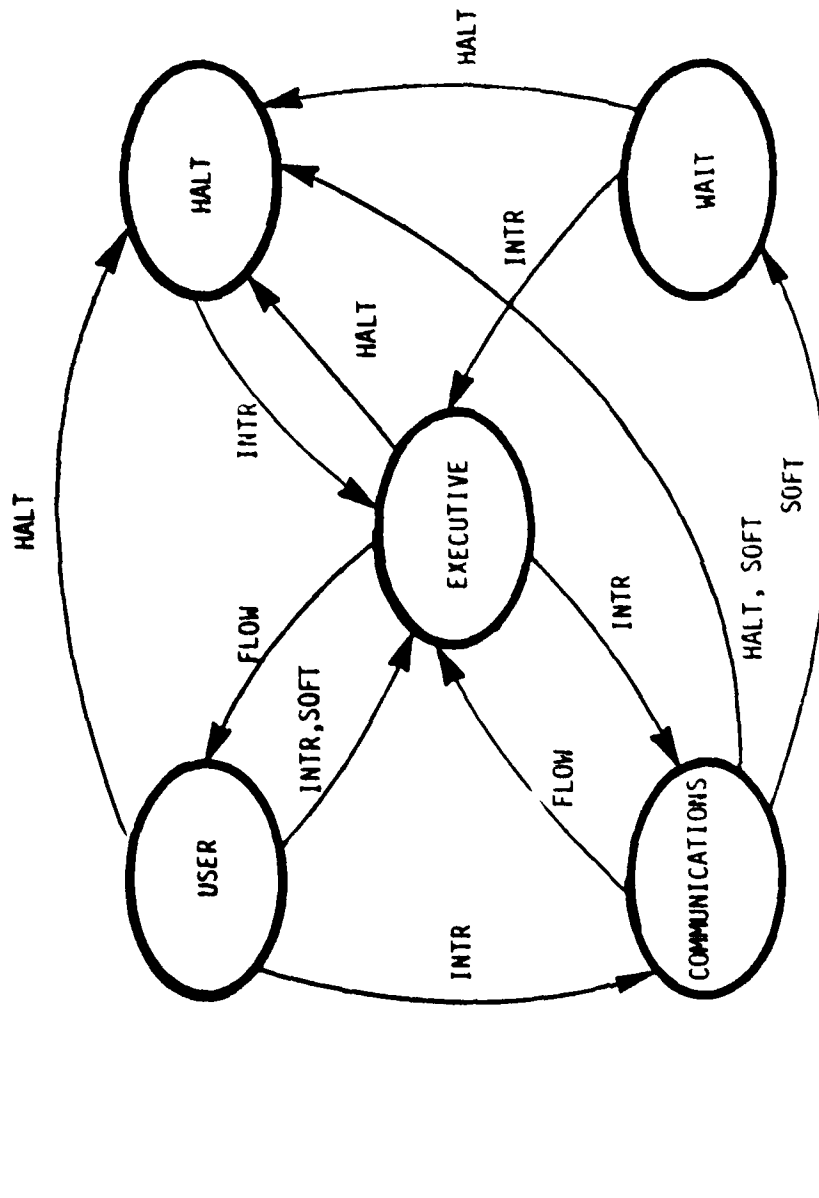
The control algorithm for the MMCS must function as both a real-time executive and as the control program for the specific tasks performed by the MMCS. The role of the executive is to provide the capability of running concurrent tasks, as described in the Introduction, and to provide the services required by the control program (the software part of the control algorithm). Thus, the control program exists, not so much as a separate entity, but as a set of parameters for the executive. In this section, several of the more important features of the control algorithm are presented.

Hardware, Software, and Firmware

The control algorithm is implemented as a function of hardware, software, and firmware (microcode). The criteria used for the partitioning of the control algorithm into hardware, software, and firmware are performance, ease and economy of implementation, and the ability to enforce the necessary rules for concurrent programs. The general design rule is to implement the fixed portion of the system in hardware and to use software for the variable or application-sensitive portion. Firmware is used in place of hardware when it is not possible or cost-effective to use a pure hardware design. Also firmware is used for those functions that have a high probability of changing as the system evolves.

The Virtual Machine Concept

Each of the microcomputer modules communicates with the rest of the system and the external world via well-defined buses. All communications within the system are handled by a virtual machine implemented in the microcomputer module firmware. The characteristics of the virtual machine are independent of the actual hardware, allowing for substantial modifications in the microcomputer hardware (thus precluding early obsolescence) without requiring changes in the control algorithm. The virtual machine has five states--HALT, WAIT, COMMUNICATIONS, EXECUTIVE, and USER--shown in the state diagram of Figure 10. The HALT state is used to take a processor module off-line for an indefinite time. The WAIT state is similar to HALT but is used to synchronize multiple units. A common control line can cause all waiting units to enter the EXECUTIVE state simultaneously. Most of the major overhead activities, such as scheduling of tasks, take place in the EXECUTIVE state. Some system resources, such as the shared memory, are available only in the EXECUTIVE



INTR: INTERRUPT
 HALT: HARDWARE HALT COMMAND
 SOFT: SOFTWARE CONTROL
 FLOW: NORMAL PROGRAM FLOW

Figure 10. Virtual Machine State Diagram

state. The USER state is used to run all of the actual application programs. If an application program wishes to make use of a restricted system resource it does so by a supervisor call (SVC) which causes the system to enter the EXECUTIVE state as long as the resource is in use. The COMMUNICATIONS state is used in the transmission of programs, control information, and other operating parameters between the control processor and the other system processors. All messages sent to a processor in the COMMUNICATIONS state are interpreted by the virtual machine. The control processor does not have to be involved with any hardware details of the module with which it is communicating. Communication with the external world by way of the I/O bus is not handled by the virtual machine. This is done directly by an application program.

Shared Memory

The MMCS has a shared memory which is available to all microcomputer modules. The shared memory, which is used to hold all system parameters and common or global variables serves as a vehicle for passing messages between processors. The shared memory is implemented as a critical region in that only one processor at a time may change the contents of a shared memory location.

Asynchronous Operation

Whenever possible the partitioned application tasks operate asynchronously with respect to the passing of variables to and from shared memory. This frees the applications programmer from the burden of dealing with scheduling accesses to parameters in shared memory for these tasks. Each module has, in local program memory, copies of all necessary shared memory variables. All references by applications programs to these variables are to the copies located in program memory. The virtual machine periodically updates the copies at a rate that is determined during program linking. The virtual machine also provides for asynchronous or event-driven updating of these variables if desired. This is illustrated by Figure 11.

Identical Hardware Modules

All of the microcomputer modules are identical in hardware and firmware. The control processor has capabilities denied to the other processors, but these capabilities are a function only of its position on the bus.

Hierarchical Organization

The control processor and the virtual machines of the other processors form a hierarchical structure. For the system as a whole the control processor presents the same external functional appearance as the virtual machines of the other modules. This allows the system to be expanded to a new layer if desired. This structure is illustrated in Figure 12.

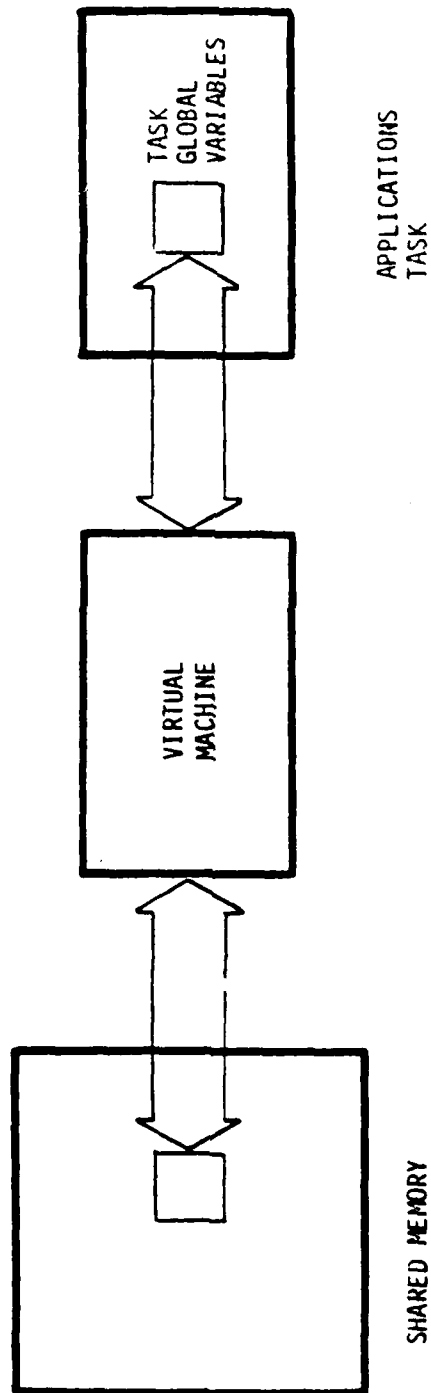


Figure 11. Role of Virtual Machine in Transfer of Task Variables

SYSTEM ARCHITECTURE

A block diagram of the system hardware is shown in Figure 13. The system consists of a control processor, I/O processors, application processors, a bus arbitration module (BAM), and a shared memory, all connected by a common communications and control bus. The computations associated with a particular system use are partitioned among the application processors. The I/O processors handle the actual interfacing of the system to external devices and may do some post or preprocessing of the data. The control processor handles the initial start-up of the system, the system diagnostics, and the overall control of the execution of the application program. The bus arbitration module handles access to the shared memory bus according to a fixed priority scheme which establishes the shared memory as a critical region.

Bus Structure

All of the system elements are linked by a common system bus--the system communication and control bus. This bus consists of two parts, a shared memory bus and a control bus. The shared memory bus is used by all processor modules to reference shared memory. Because the shared memory is considered to be a critical region, the shared memory bus can be used by only one processor at a given time. The processor modules acquire the bus according to a priority scheme to be described later. Data transfers on the shared memory bus are synchronous once the bus is acquired. This gives the highest possible throughput but requires that the memory response be able to match the processor cycle time. A wait line can be provided to allow the use of slower memory if desired.

The control bus is a single-master bus used by the control processor for all system control functions. While all processor modules are identical, only one processor can be master of the control bus. This processor is designated by placing it in the slot closest to the bus arbitration module. One pin of this connector is grounded, enabling the processor to function as a master of the control bus. With this one exception, all bus locations are identical. The control bus is an asynchronous bus and all communications are handled on a request/grant basis.

Shared Memory Bus Priority Arbitration

The signal structure of the shared memory bus arbitration lines is shown in Figure 14. There are four separate functions: the bus-request lines ($BR_0 - BR_{(N-1)}$), the bus-grant lines ($BG_0 - BG_{(N-1)}$), the grant-acknowledge line (GACK), and the bus-locked (LOCK) line. The number of bus-request lines and bus-grant lines corresponds to the number of priority levels used. Eight

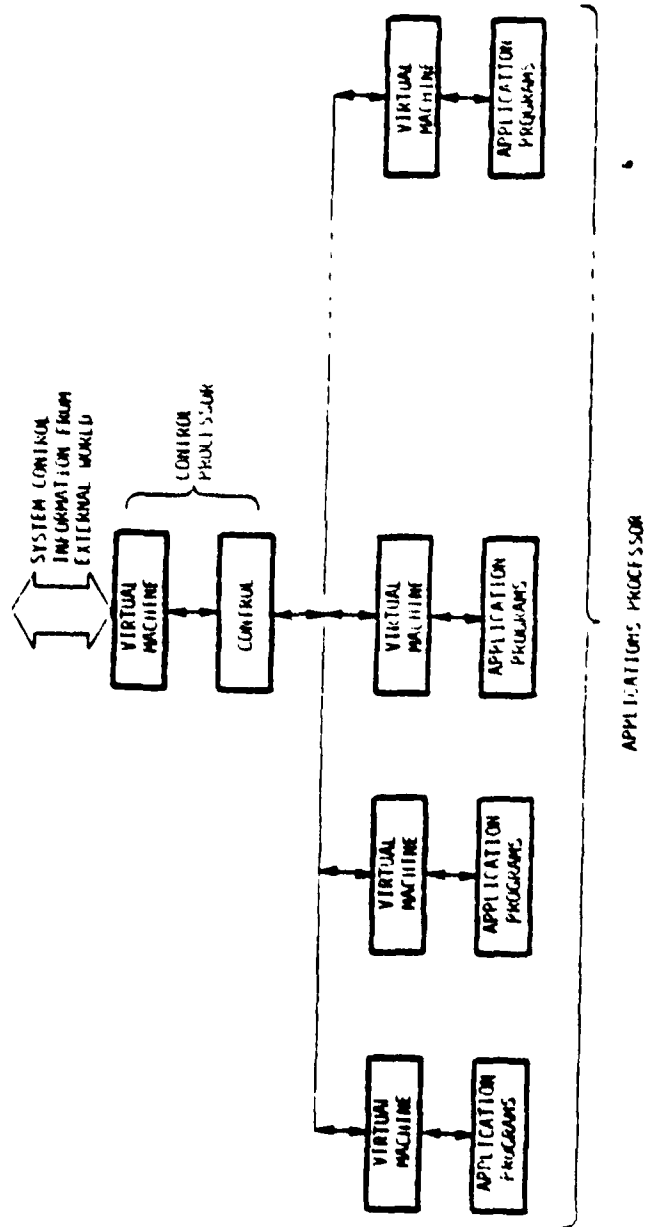


Figure 12. Hierarchical Organization of MMCS

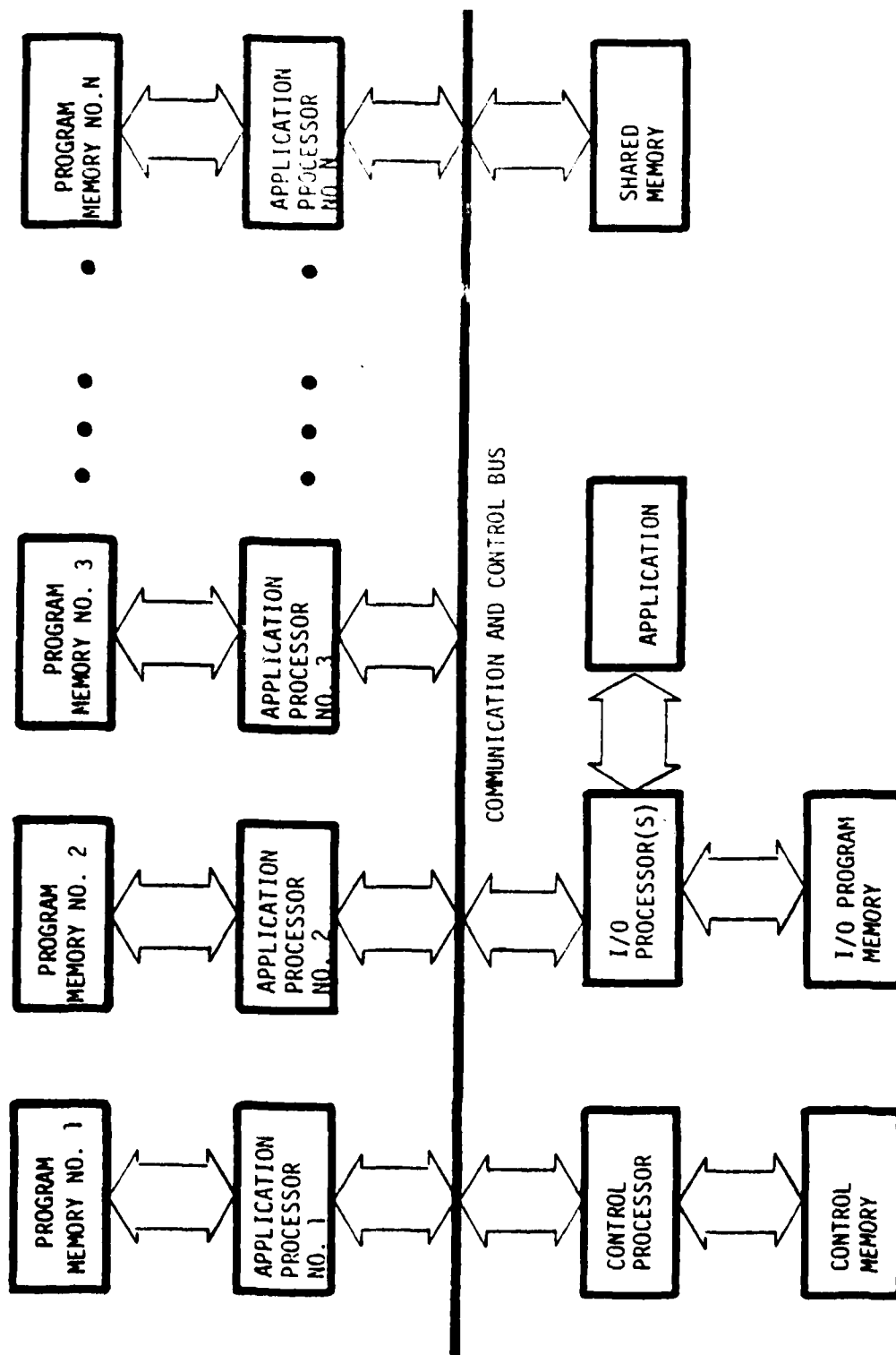


Figure 13. MMCS Block Diagram

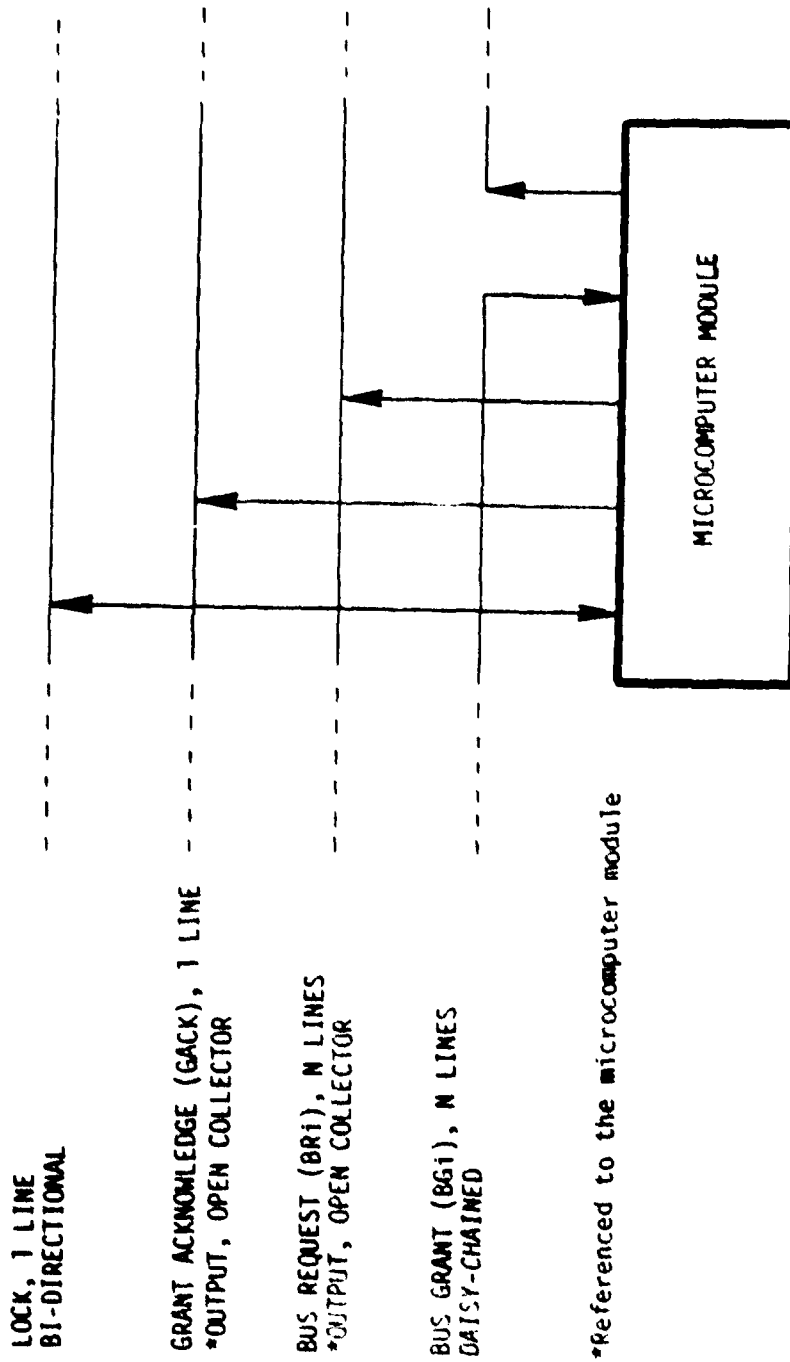


Figure 14. Signal Structure of the Bus Arbitration Lines

lines are used in the design example and this appears to be the optimum choice. However, the number can be readily changed if required. The daisy-chain connection of the bus-grant lines causes the priority of a given module to be a function of position on the bus. By blocking the propagation of a bus-grant signal, a module can deny a "down-stream" module access to the bus.

Access to the bus is handled by circuitry on the processor modules and by a bus arbitration module (BAM). The BAM is connected at one physical end of the bus. The position next to the BAM is the highest priority for each of the request-grant line pairs and is used by the control processor. Operation of the arbitration scheme is as follows: A processor module which desires access to the bus will assert one of the bus-request lines, the choice of request line being determined by the nature of the transfer. Selection of the request line is handled by the virtual machine, not by an application task. If no higher priority is asserted, and if no module located closer to the BAM on the same line has requested the bus, then the BAM will assert the corresponding grant line and the grant will not be blocked before arriving at the processor. (The effects of processor position and choice of bus-request line on priority are illustrated in Figure 15.) Upon receiving the bus request the processor asserts GACK and has acquired the right to be the next user of the bus. As soon as the present bus user (if any) clears LOCK, the processor asserts LOCK and proceeds to use the bus as desired. When it has completed bus activity, it first clears GACK, thus enabling the BAM to assert the present highest bus-grant line, then clears LOCK, thus giving up the bus. If possible the processor should clear GACK in anticipation of giving up the bus, allowing the next arbitration sequence to proceed early, giving better bus utilization.

A flowchart of the operation of the BAM is shown in Figure 16. While the bus arbitration sequence itself is asynchronous, the BAM operates as a synchronous finite-state machine clocked by the processor clock. The state diagram for the BAM is shown in Figure 17. A block diagram of the BAM logic circuitry for $N = 8$ (i.e., 8 bus grant/request pairs) is shown in Figure 18. Most of the BAM logic is implemented with three MSI packages. The logic for the next state signals (shown) and the output decode is SSI (three 3-input NAND gates). Two D type flip-flops (1 package) are used to provide the state variables. The total package count for the BAM (for $N = 8$) is seven 16-pin packages.

A flowchart of operation of processor module portion of the bus arbitration logic is shown in Figure 19. While this logic is also a finite-state machine, it differs from the BAM in that it is implemented in software and firmware as well as hardware.

The bus-request and bus-grant lines are assigned by function. In general, the write requests have a higher priority than the read requests. All N-cycle

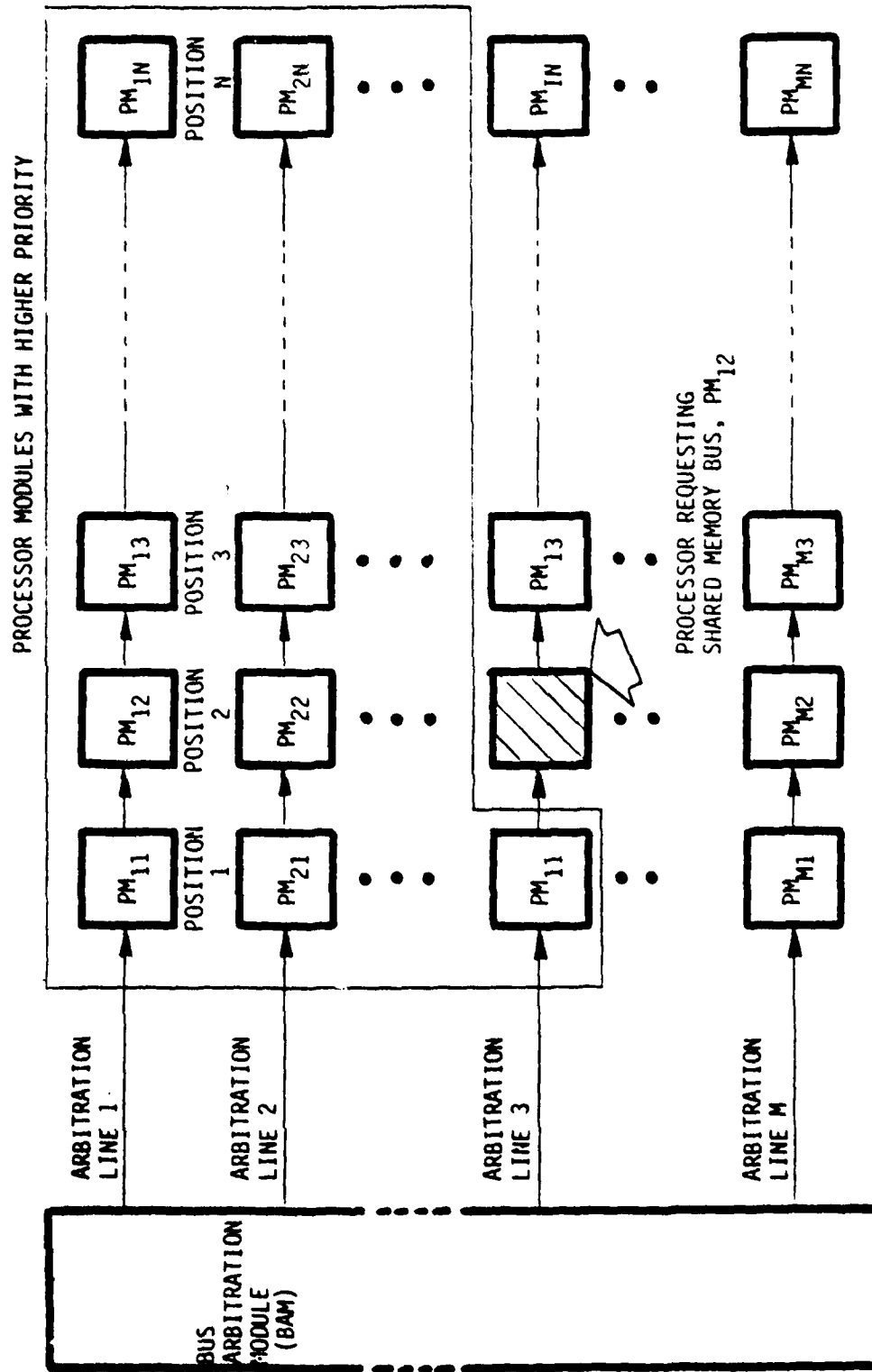


Figure 15. Operation of Shared Memory Priority Scheme

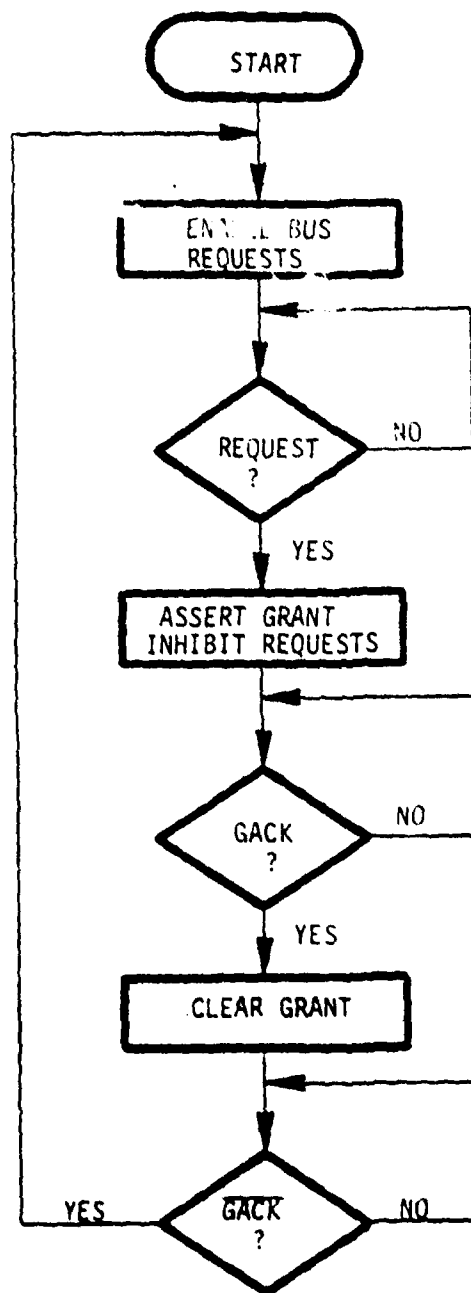


Figure 16. Flowchart of BAM Operation

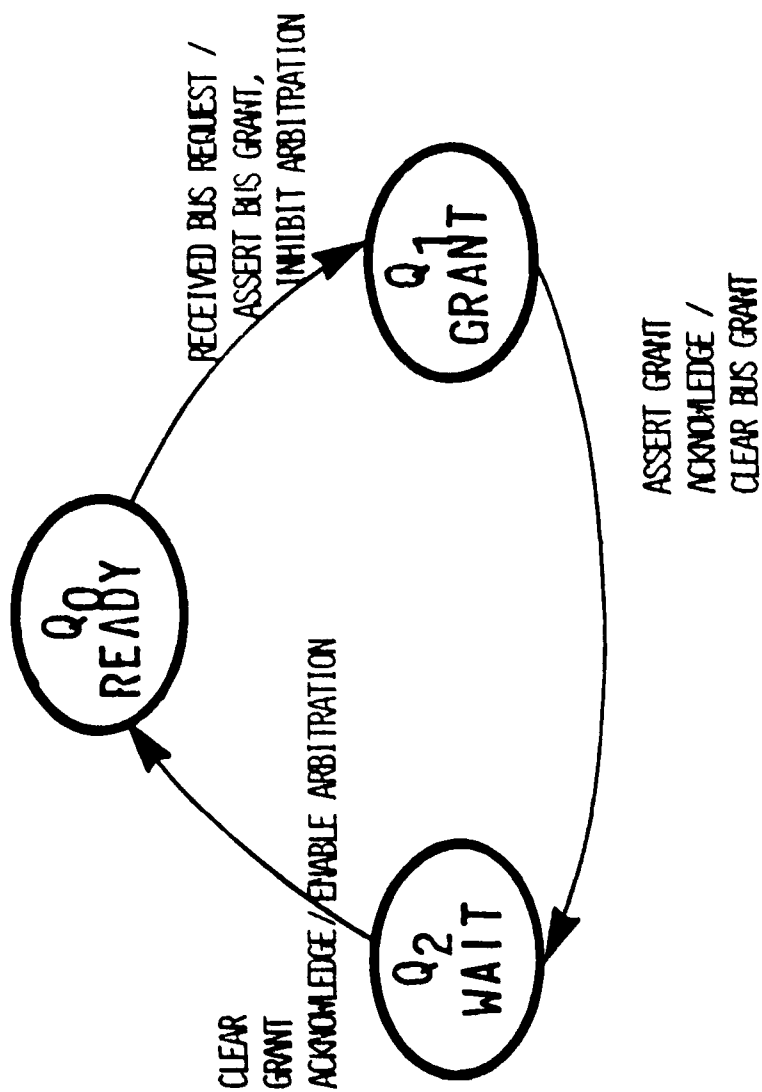


Figure 17. BAM State Diagram

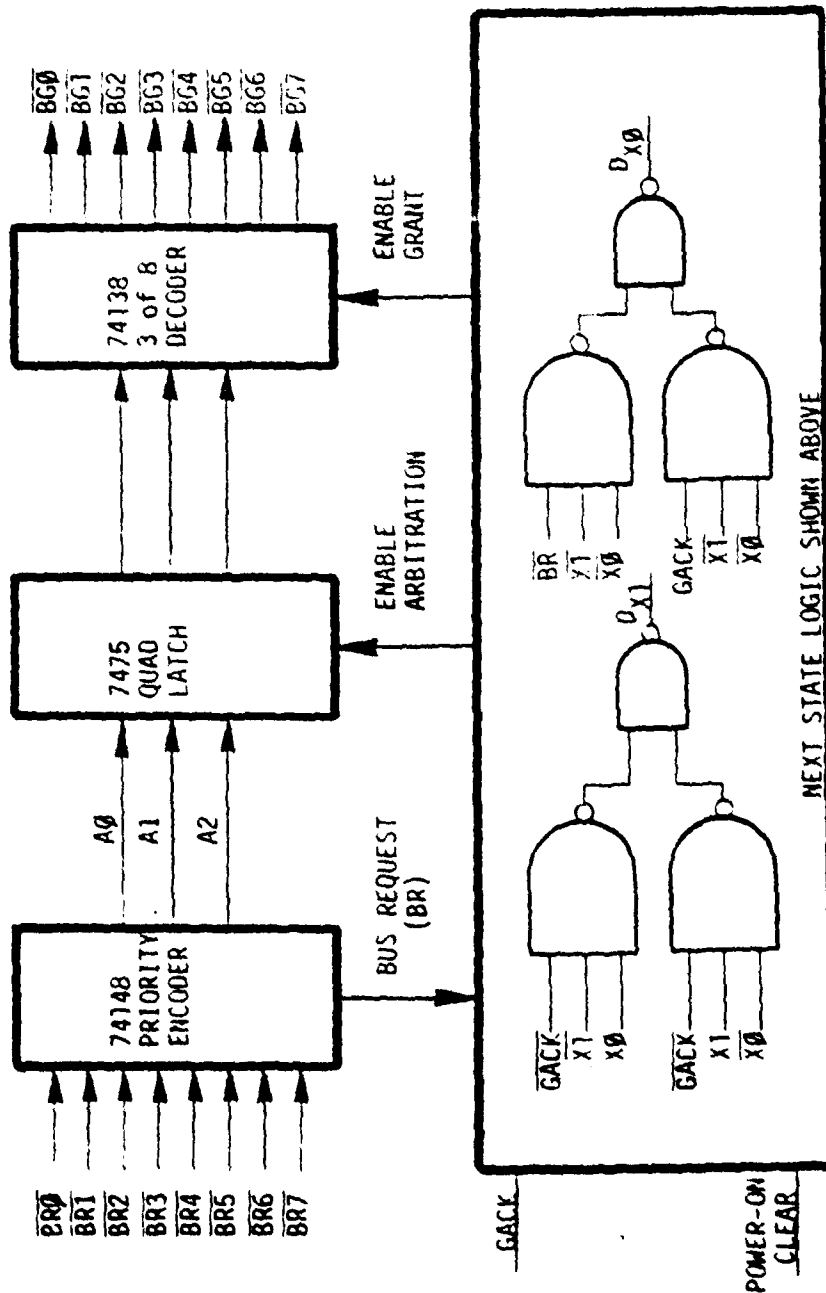


Figure 18. Simplified Block Diagram of BAM Logic

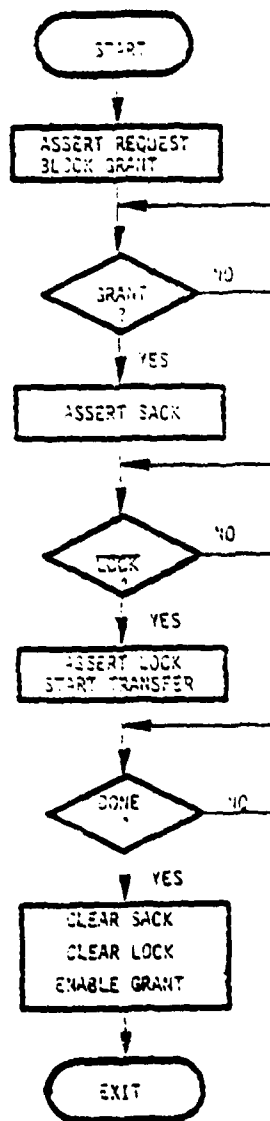


Figure 19. Flowchart for Operation of Processor Module Priority Logic

requests are scheduled at a higher priority than K-cycle requests. A special priority request line is provided for exceptions. The highest priority is reserved for the control processor. Sample assignments, for $N = 8$, are given in Table 2.

The Distributed Cache Shared Memory

The ultimate limiting factor on the performance of the MMCS is shared memory bus bandwidth. The computational power of the system can be increased by adding additional processor modules but this has the effect of increasing the burden placed on the shared memory bus. Examination of the activity on the shared memory bus suggests that:

- a) The number of memory reads must at least equal to the number of writes since the shared memory is used for holding and passing parameters.
- b) Many system variables will not be subject to frequent updates and will be read many more times than they are written.
- c) Some parameters, produced by a single processor (or obtained through external I/O), will be passed to several other processors.

The net effect is that the number of read cycles on the shared memory bus will probably exceed the number of writes by a substantial amount, the exact amount being a function of the application and the partitioning scheme. This points to a technique which could substantially reduce the amount of traffic on the shared memory bus, the distributed-cache shared memory (DCSM).

The DCSM increases the effective bandwidth of the shared memory bus by two mechanisms:

- a) It provides for a "broadcast" write operation whereby information can be written into several processors simultaneously.
- b) Parameters which are not updated on a regular basis may be stored "locally" at the processors where they are used.

The basic concept of the DCSM is that portions of the shared memory are duplicated at some, or all, of the processor modules. This is illustrated in Figure 20 which shows two separate segments of shared memory duplicated in certain processor modules. It is necessary that the DCSM operate in such a way that shared memory retains a single unique address space. This is accomplished by requiring that all writes to shared memory be global, updating all local cache memories at once.

The DCSM adds a new arbitration problem to the system, the situation where both a local read and a global write occur for the same block of memory simultaneously. One solution is to have a busy flag for each local cache

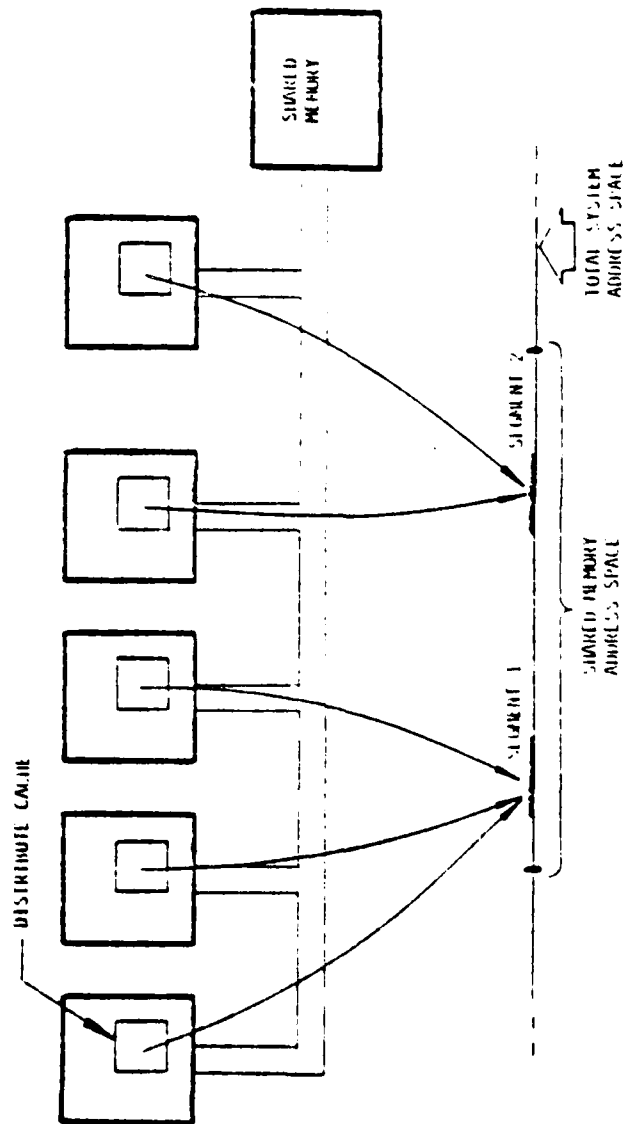


Figure 20. Illustration of Distributed Cache Allocation

TABLE 2. BUS REQUEST ASSIGNMENTS

| <u>BUS REQUEST</u> | <u>PRIORITY</u> | <u>ASSIGNMENT</u> |
|--------------------|-----------------|---|
| BR0 | 0 | Used by the control processor for both read and write operations. |
| BR1 | 1 | ATM Priority Write. Used by ATM to write control information into shared memory. |
| BR2 | 2 | ATM Priority Read. Used by ATM to obtain control information from shared memory. |
| BR3 | 3 | N-Cycle Write. |
| BR4 | 4 | N-Cycle Read. |
| BR5 | 5 | K-Cycle Write. |
| BR6 | 6 | K-Cycle Read. |
| BR7 | 7 | Distributed Cache Read. Used to obtain bus to read cache memory. More than one processor may be granted this request at one time. |

and give the global activity the highest priority. This has the disadvantage of requiring additional hardware and adding overhead to each shared memory cycle. A second solution is to dedicate one bus request line to the DCSM and allow all local reads to be handled in parallel. This solution has the disadvantage of a longer latency time, but adds minimal hardware and no overhead. The nature of the type of variables stored in the cache reduces the disadvantage of the higher latency time.

The Application Task Manager

A computer operating in a multi-tasking environment requires some form of real-time executive to handle allocation of system resources. In the Introduction, a number of the desired attributes of the executive to be used in the multiple microcomputer system are detailed. These attributes, along with the special requirements of a multiple-processor system have been used in the design of the applications task manager (ATM). The ATM handles all real-time resource allocations and all real-time intertask communications for each of the processors. In addition, the virtual machine which handles all communications between processors is implemented by the ATM. This communications-oriented portion of the ATM is the major difference between it and a single-processor executive. The role of ATM in the system is shown in Figure 21. The main control program runs under the ATM of the control processor and directs the ATM's of the various applications processors. These ATM's, in turn, control the application programs executing in their respective processors. External control of the entire system is handled through the ATM of the control processor. The ATM is described in more detail in a later section of the report.

The Microcomputer Modules

A microcomputer module of the MMCS and all of its major interfaces are shown in Figure 22. On the system side, the module interfaces to the shared memory bus and the control bus through interface logic. The interface to the shared memory bus also includes the arbitration logic. The control bus interface is less complicated since the control bus needs no arbitration logic. The status and control register in each processor contains the control information for that module. The control processor treats these registers as a sequence of addresses on the control bus. Some of the bits are modified by the control processor and are used to control the activities of each processor module. Other bits are read by the control processor to obtain the status of the processor modules.

THE APPLICATIONS TASK MANAGER

The ATM controls the execution of all application programs. It is an integral part of each processor module and of the MMCS architecture. The ATM is implemented in both microcode and native code (which may be compiler generated).

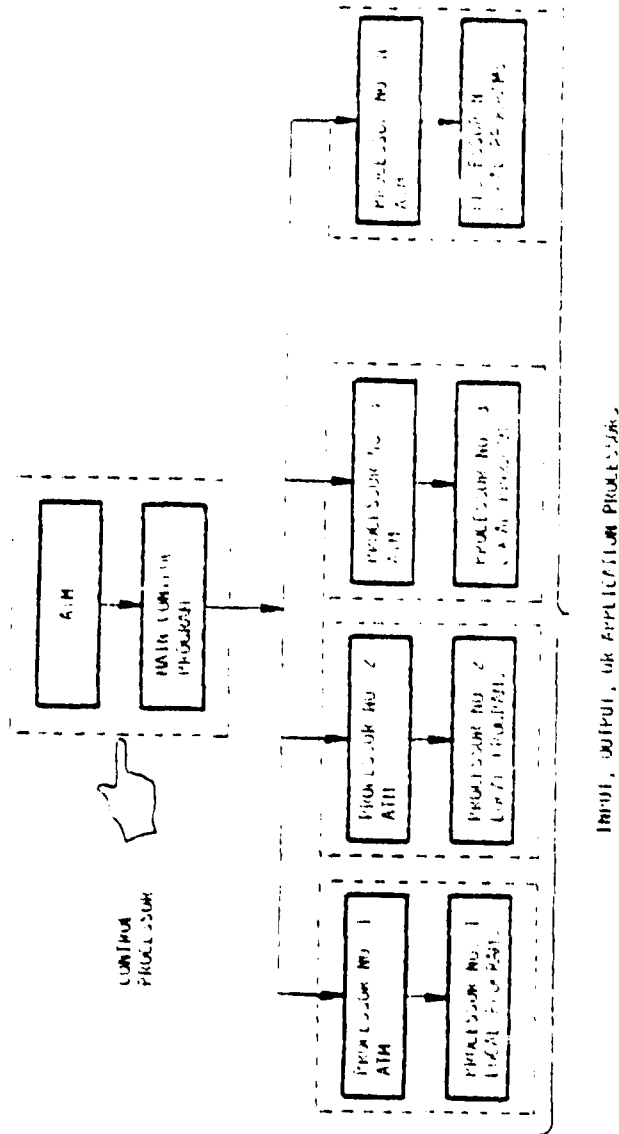


Figure 21. Role of ATM in MMCS

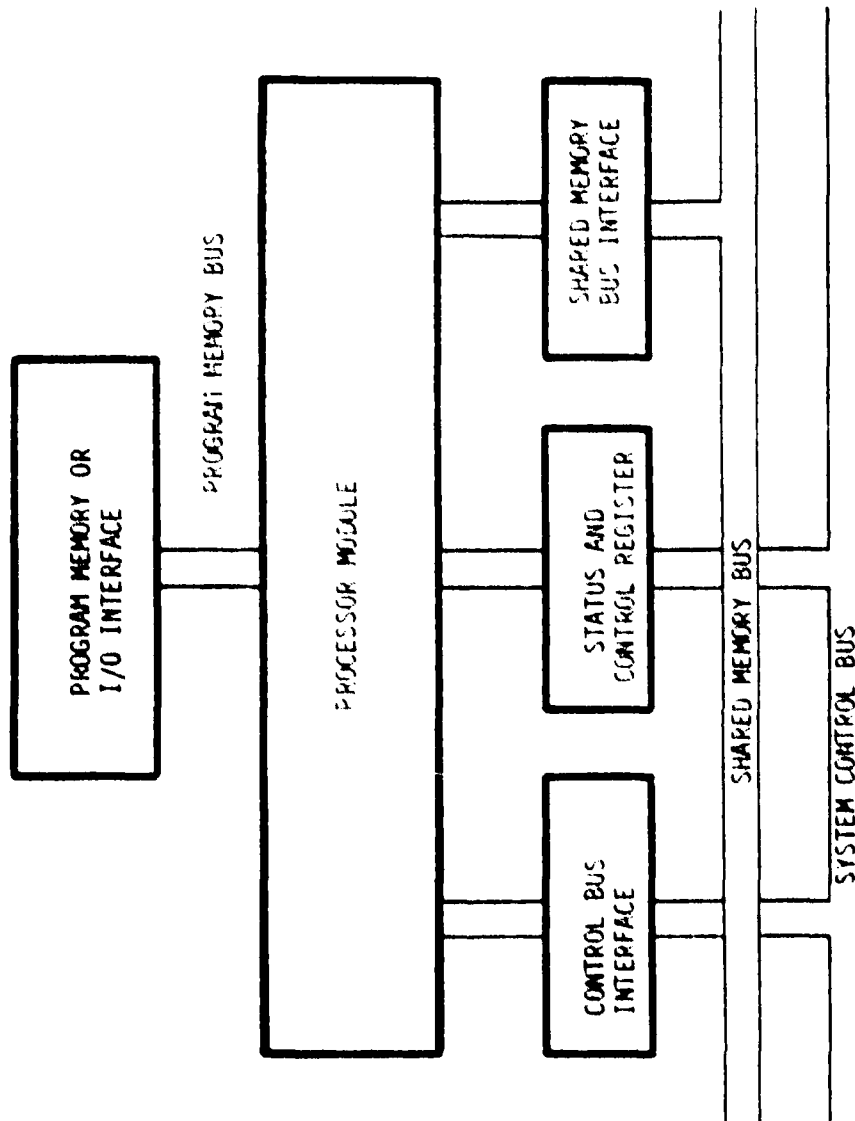


Figure 22. Processor Module and Bus Interface

ATM Functions

The main functions of the ATM are to:

- a) Handle allocation of all shared resources.
- b) Handle scheduling of all tasks.
- c) Implement the virtual machine used for system communication.
- d) Provide a structure for isolating applications programs from hardware details so that changes in the hardware do not require the applications programs to be rewritten.
- e) Provide the means by which the system may be initialized after a cold start.
- f) Provide diagnostic tools for debugging programs.

ATM Operation

The discussion of ATM operation given here is sufficiently general so as to be independent of differences in system hardware. The actual implementation must take into account the capabilities of the actual hardware. A subsequent section outlines the system hardware requirements for an efficient implementation of the ATM.

Basic Concepts. A simplified flowchart of the operation of the ATM is shown in Figure 23. Entry into the ATM is always by an interrupt or trap (or software interrupt). Upon entry to ATM the initial activities are to determine the type of interrupt and to set the ATM flags and queue pointers appropriately. Normally control is passed directly to the scheduler which assigns a starting time to the desired task. However, under some conditions an exceptional task may be activated. Exceptional tasks include:

- a) the supervisor call (SVC) handler,
- b) the error handler,
- c) the cold-start initialization routine, and
- d) the communications state of the virtual machine.

When the exceptional task is complete, control is passed to the scheduler as in a normal interrupt. Exit from the ATM is always to the scheduled application task. In the event that no applications task is currently ready to run, a diagnostic task can be scheduled. No specific provision is made for implementing a background task since the same effect is achieved by scheduling a task at the lowest priority with unlimited time. The diagnostic task, if present, would be handled in this manner. In the event that no task is ready to run, the ATM will schedule a null job. A complete flowchart of the operation of the ATM is given in Figure 24.

Several of the key features of the ATM may be observed from the flowcharts of Figures 23 and 24, in particular:

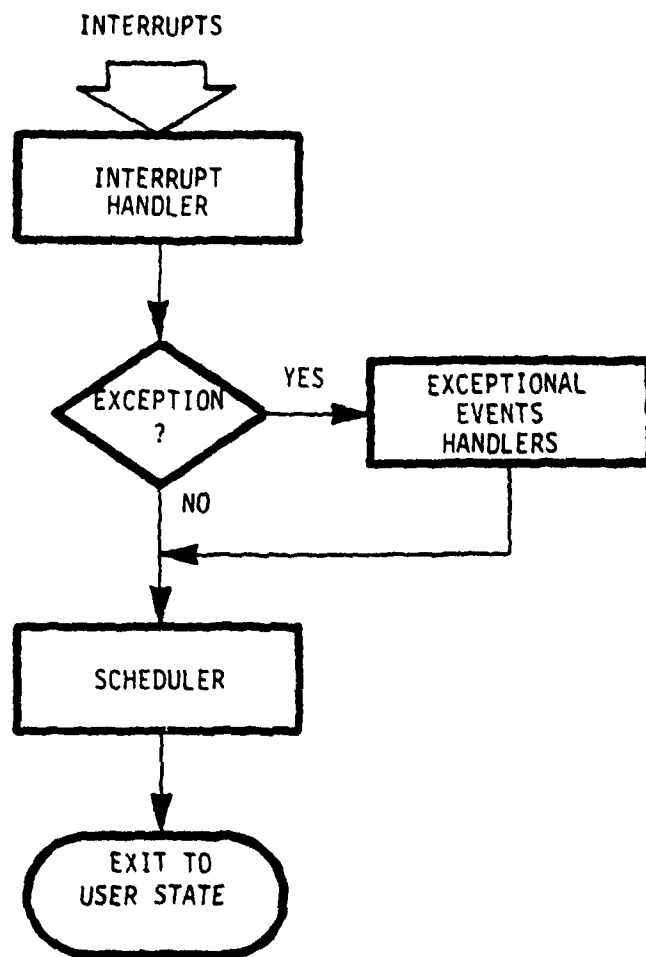


Figure 23. Simplified Flowchart of ATM Operation

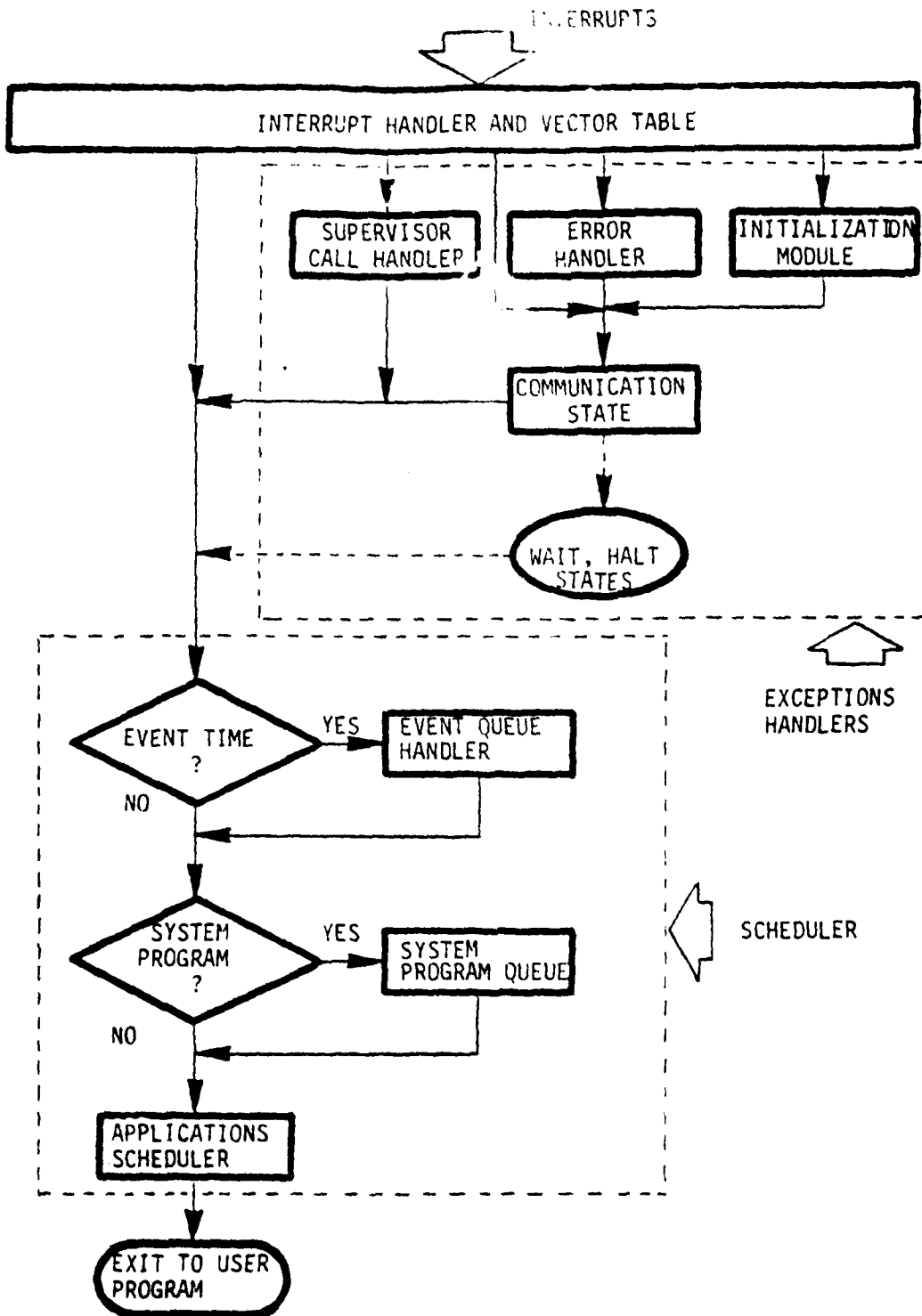


Figure 24. Complete ATM Flowchart

- a) While ATM is activated by interrupts, these interrupts do not cause direct execution of any tasks. Instead they set certain ATM flags and which allow passage of control to the scheduler. Thus, the scheduler always selects the task to be executed, regardless of the interrupt type.
- b) Exit from ATM is always to the task selected by the scheduler.

MMCS Interrupts. The processor modules of the MMCS have six classes of interrupts, of which five are handled by the ATM. The six classes are defined below:

CLASS 1) SYSTEM LEVEL INTERRUPTS

These interrupts are used for events which affect the entire system. They are part of the system bus structure and cannot be masked by the processors. They may originate from the control processor or from the external world. Examples of the use of system interrupts are the power-fail routine and the system "freeze" function.

CLASS 2) COMMUNICATIONS REQUEST INTERRUPTS

These interrupts cause the processor module to go to the communications state and acknowledge the request. The interrupting device may then initiate the desired exchange.

CLASS 3) ERROR TRAPS

These interrupts occur as the result of an attempt to violate system policy. They cause an error handler to be started.

CLASS 4) ATM TIMER INTERRUPTS

These interrupts are the result of an ATM timer and are used by the ATM for scheduling purposes. They are the normal cause of entry to ATM.

CLASS 5) ATM ASSIGNABLE INTERRUPTS

These interrupts cause a specific task to be flagged to run and can be assigned by ATM.

CLASS 6) APPLICATIONS INTERRUPTS

These interrupts are controlled by the applications programs.

They do not cause entry to ATM and are masked whenever ATM is active. A given interrupt is associated with a specific task and is unmasked only when that task is active. Assignment of these interrupts is done by a supervisor call to ATM.

The priority of the interrupts is highest for class 1 and lowest for class 6. Within a given class the interrupts are prioritized if there is more than one.

The Interrupt Handler and Vector Table. The interrupt handler and vector table handle class 1 through class 5 interrupts and the overhead associated with entry into ATM. If possible, each interrupt is assigned a vector in the vector table for the fastest possible response. The class 1, class 2, and class 3 interrupts, which cause specific exceptional tasks to be activated, are always serviced immediately. The class 4 interrupts, whose primary purpose is to activate the scheduler, do not cause specific tasks to run but can affect flags in the scheduler. The class 5 interrupts flag specific tasks to be run and activate the scheduler.

The chief overhead task is to handle the possibility of the ATM being interrupted. Most interrupts are disabled while the ATM is active but some class 1, class 2, and class 3 interrupts are never masked. If the ATM is interrupted, a flag is set so that the previous activity is completed before exiting to an application program.

Supervisor Call Handler. The virtual machine implemented by the ATM provides a group of services to the application tasks. These services are in the form of new instructions at the task level. The mechanism by which they are invoked is the supervisor call (SVC). A task issues a SVC by use of a software interrupt (or trap) instruction followed by the code for the desired supervisor call. The ATM vectors this request to the supervisor call handler where it is decoded and executed. At completion of the SVC, the ATM is exited through the scheduler.

The Error Handler. The error handler identifies the error code and prepares a message to be passed to the communications state. Other information, such as the machine state or the identity of the active task, is supplied where appropriate. This information is passed to the control processor. The task may or may not be restarted, depending upon the severity of the error. Control is always passed to the scheduler after leaving the communications state.

Initialization Routine. Application of reset or cold-start interrupt causes the initialization program to be run. After the system is initialized control is passed to the communications state. The ATM remains in this state until it receives a command from the control processor (or, in the case of the control processor, from the external world).

The COMMUNICATIONS, HALT, and WAIT States. These three states have previously been described as a part of the virtual machine. A processor module enters the COMMUNICATIONS state upon receiving a class 2 interrupt. Typically two class 2 interrupts are provided, one maskable and one non-maskable. The maskable interrupt is used for routine requests and does not cause the processor to suspend a critical task. When the processor enters the COMMUNICATIONS state it asserts the output handshake line to indicate that it is ready. If the control processor initiated the request, it will then send the first word of the message and assert the input handshake line which clears the output handshake line. When the processor reads the word it again asserts the output handshake line, clearing the input line in the process. This sequence continues until entire message has been sent. If the processor is the initiator, it waits until the input line is asserted and then outputs a word to the control bus. Again, the process continues until the entire message has been transmitted.

The messages exchanged by processor modules in the COMMUNICATIONS state have the format opcode, operands, terminator and are interpreted by the virtual machine. The COMMUNICATIONS state operations are given in APPENDIX A.

The HALT and WAIT states are normally entered as a result of a command sent as a part of a message. However, the status and control register of each processor module also contains a hardware halt bit.

Scheduler. The scheduler handles all tasks other than exceptional tasks. It also handles all time-dependent functions for the processor. The scheduler is described in detail in a section to follow.

ATM Time-Keeping Functions. During normal operation the ATM maintains three time-dependent activities. These are the cycle-clock counter, the task timer, and the event-alarm timer. The cycle clock is the external timing signal fed to each module for synchronization. The cycle clock sets the rate at which shared memory read/write cycles may occur. Because a given processor module may not have to access shared memory every cycle, a counter is used to provide an interrupt at the desired intervals. After each interrupt, the cycle counter is first loaded with the number of cycles before the next interrupt and then enabled. It is driven by the cycle clock which is available on the system bus.

The event timer and the task timer are both driven by the system clock, which is also available on the system bus. The system clock is the reference for all time dependent activities. The period of the system clock sets the lower limit for the time resolution in the MMCS. The event timer is used to generate an interrupt at the time of the next scheduled event. This interrupt also sets a flag which informs ATM that the event queue--a queue of all scheduled events--should be checked. Each time the event queue is checked, the present event is removed and the corresponding task activated. The event timer is then loaded with the time available until the next scheduled event. If desired, the functions of the cycle-clock counter may be implemented by use of the event timer. This is a more general technique but may cause slightly more overhead.

The task timer is used to control the amount of time consumed by an active task. When a task is started its associated time limit (if any) is loaded into the task timer. The task timer counts down as long as the system is in the user mode (it is inhibited while the ATM is active). If a task is swapped out (pre-empted) before its time is up, the time remaining is saved and restored when the task is restarted. The task timer may be an actual timer or it may be implemented as an event on the event timer.

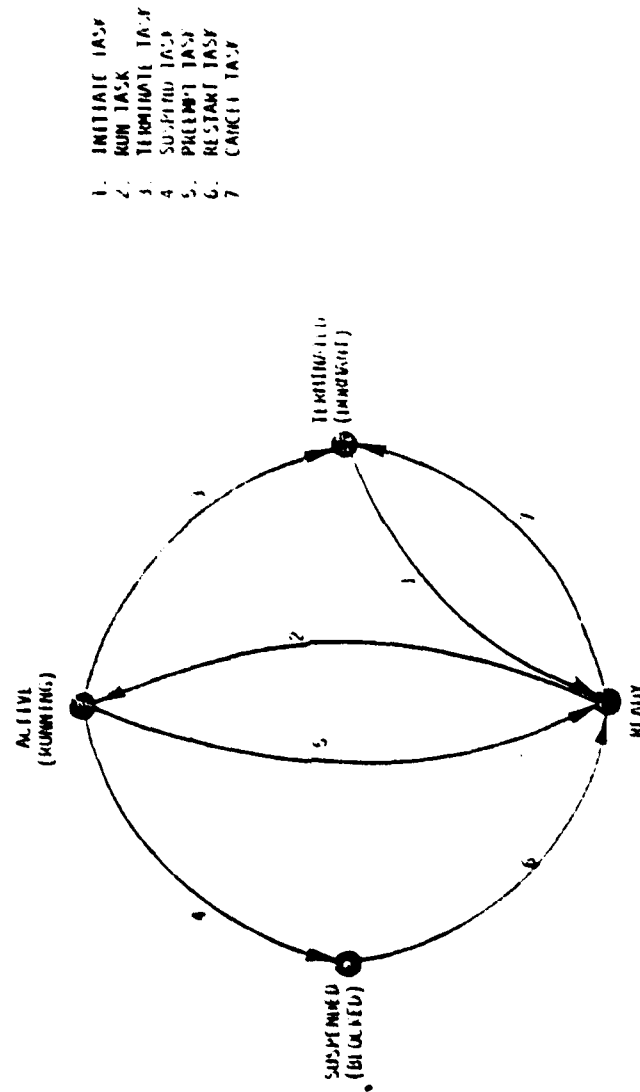
Application and System Program Tasks ATM supports two types of tasks, application tasks and system tasks. The application tasks, as the name suggest, support the application assigned to the processor module. System tasks (or, equivalently, system programs) are provided as a means of implementing executive functions that are matched to a particular application. System programs have the following characteristics:

- a) System programs run to completion. They are not preempted except by their own time limits.
- b) All system programs that wish to run do so when ATM is entered.
- c) System programs have access to all processor resources.
- d) The highest priority system program (the one that runs first in view of point) can be flagged to run by the cycle clock counter interrupt.
- e) A system program can be caused to run by a class 5 (assignable) interrupt.

System programs are allowed in order to make the ATM as extensible as possible. However, these are to be used as application-dependent parts of ATM, not as higher-priority application programs.

Tasks States. The possible task states are ACTIVE (RUNNING), READY, SUSPENDED, (BLOCKED) AND TERMINATED (DORMANT). An active task is the one which a processor is presently executing; a given processor module can have only one ready task. A terminated task is one which either has never been scheduled to run or has run to completion. A processor module can have more than one terminated task. A suspended task is one which has previously been running and has terminated itself when a specific event occurred. For example, a task may issue a supervisor call to pause for a time interval or to pause until specific conditions have been satisfied. When the conditions for restarting the task are satisfied, the task is placed in the ready-state and executed as soon as it becomes the highest priority task. A processor module may have more than one suspended task. A task state diagram is shown in Figure 25.

The ATM Scheduling Mechanism. ATM uses a single-level dynamic priority assignment and preemptive scheduling with resumption. This implies that it is possible to interrupt the execution of a task to run a higher-priority task. The interrupted task can be continued later when the higher-priority task ter-



1. INITIAL TASK
2. RUN TASK
3. TERMINATE TASK
4. SUSPEND TASK
5. PREPARE TASK
6. RESTART TASK
7. CANCEL TASK

Figure 25. Task State Diagram

minutes. When more than one task wishes to run, the highest-priority task is chosen. If two tasks have the same priority, they are executed in the order of their requests. The priority of a task can be changed after the task is created and loaded into the processor module. Normally, this would be done by a higher priority task rather than the task involved. ATM provides a supervisor call to modify task priority but never changes a priority without external direction because the priority of a task is considered to be a user-controlled policy and not a system function.

The Scheduler

A flowchart of the scheduler portion of the ATM is given in Figure 26. The three main components of the scheduler are the event queue handler, the system program queue, and the application program scheduler.

The Event Queue Handler. The event queue is the mechanism used to handle all events scheduled to occur either at some specific time or after an elapsed time. The entries in the event queue consist of a time and a pointer to a task. All times are absolute, the relative times having been added to the present time at entry. The entries are sorted by value with the "top" of the queue containing the first time. When the event timer interrupt occurs, the event queue handler is flagged to run by the interrupt handler. When the scheduler is entered, the task pointed to by the entry at the top of the queue is activated and this entry is removed from the queue. This does not mean that the task begins execution at this time, but only that it is able to compete for processor time. It executes immediately only if it is the highest priority ready task.

The System Program Queue. If any system program is flagged to run, the system program queue is searched and all ready programs are run. Since all system programs run, the use of priority does not have much meaning. However, there is a precedence in that the system programs are run in a well-defined order.

The Application Task Scheduler. The application task scheduler compares the priority of the currently active application task with the highest priority task which is presently ready to run. The higher priority task is scheduled to run and control is passed to that task. If no task is presently ready to run then control remains with ATM in a "wait for interrupt" mode, since the interrupt would be the only mode by which a task could be scheduled under these circumstances. This is the null job mentioned earlier.

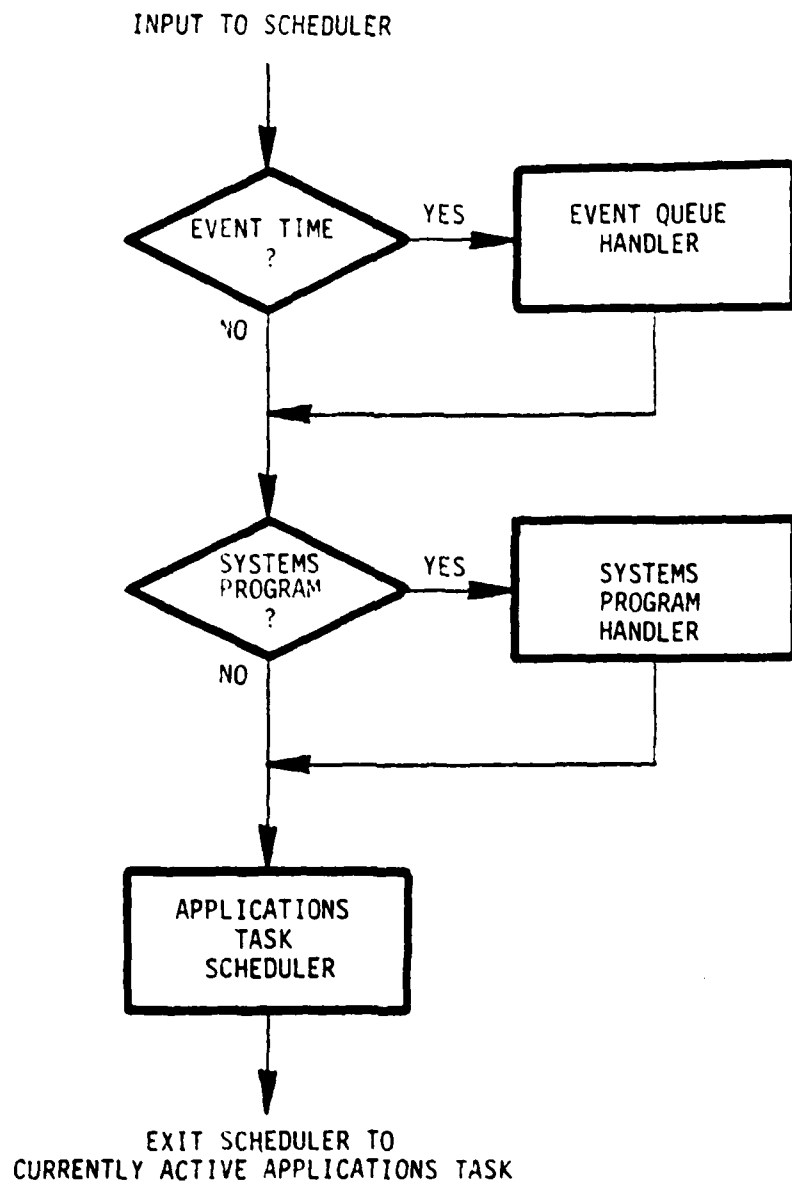


Figure 26. Flowchart of Scheduler Portion of ATM

ATM Data Structures

There are five major data structures in the ATM, as follows:

- a) Task Control Blocks (TCB)
- b) Jobs Queue
- c) Event Queue
- d) System Program Queue
- e) Ready Task Queue (RTQ)

The Task Control Block. All of the information which the system needs about a task is contained in a group of parameters called a task control block (TCB). The TCB contains the following parameters: (illustrated in Figure 27):

| <u>ENTRY</u> | <u>SIZE</u> |
|--------------------------|-------------|
| a) ID Tag | 1 byte |
| b) Priority | 1 byte |
| c) State Flags | 1 byte |
| d) Time Limit | Variable |
| e) Time Limit Storage | Variable |
| f) Starting Address | Variable |
| g) Ending Address | Variable |
| h) Stack Pointer | Variable |
| i) Stack Pointer Storage | Variable |
| j) RTQ Pointer | Variable |

The ID tag is a number from 0 to 255 used to identify a particular task. The priority is a number between 0 and 255 (0-highest, 255-lowest) used to determine scheduling precedence. The ID tag must be unique, but the same priority may be assigned to more than one job. Jobs of equal priority are serviced in the same order that they are entered. The state flags denote the state of the task at any given time. In addition some of the state flags are used to provide inputs to the scheduler. The state flag assignments are shown in Figure 28. The time limit is the maximum amount of time the task is allowed to run each time it is scheduled. This value may be changed by a supervisor call. A value of zero is interpreted as unlimited time. A second location is provided to store the elapsed time if the task is interrupted. The starting and ending addresses indicate the limits of the memory occupied by the task. The stack pointer is the initial value of the stack when the task is started. The second location is used to store the stack pointer if the task is interrupted. The RTQ pointer gives the location of the task in the ready task queue.

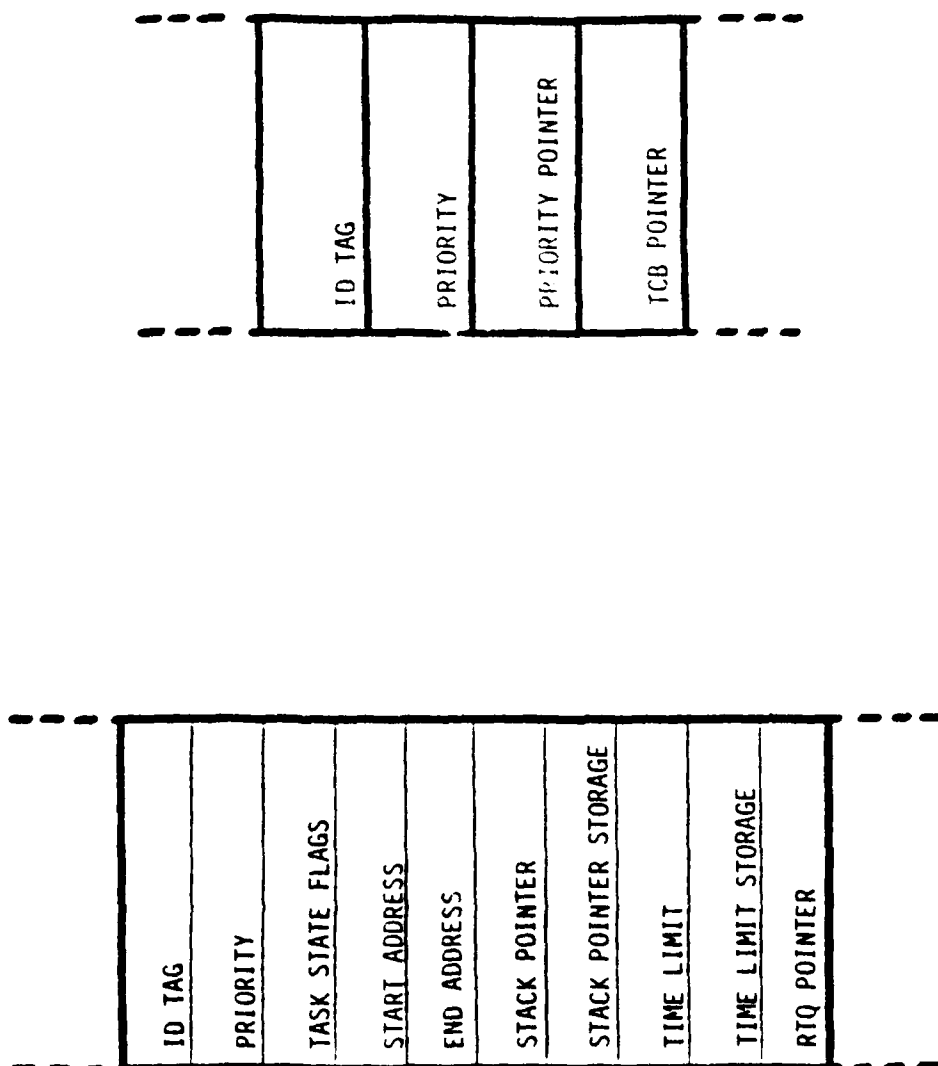


Figure 27. Task Control Block

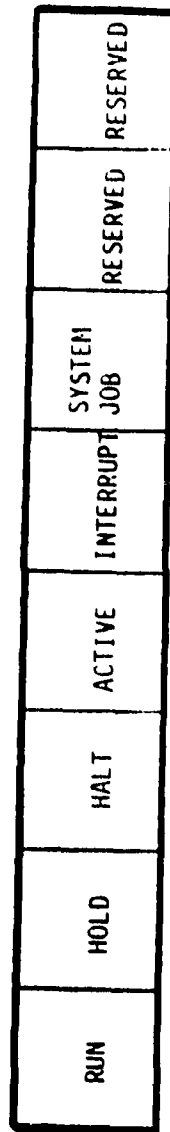


Figure 28. Task Control Block State Flags

The Jobs Queue. The jobs queue is used to find the TCB or RTQ entry (the ready task queue is described in a following section) of a given job. It is organized by ID tag and contains the following items for each entry

- a) ID tag
- b) Priority
- c) RTQ pointer
- d) TCB pointer

The jobs queue is relatively static so changes are made by physically adding or deleting entries. The ATM is designed such that the jobs queue could be changed to a linked structure later if necessary.

The Event Queue. Entries in the event queue are keyed to the event time, which is also given as an absolute time referenced to the system time base. The event queue, shown in Figure 29, consists of a linked list of entry blocks, a stack containing pointers to all empty blocks, a pointer to the current top of the queue (next event), and pointers delimiting the area of memory containing the entries. Each entry in the queue contains the following parameters:

- a) The event time
- b) An opcode defining the nature of the event
- c) Operands as necessary
- d) A forward pointer
- e) A backward pointer.

The linked-list structure is used to reduce the time necessary to search, sort and modify the queue. However, as entries are added and deleted the queue becomes sparse due to the embedded empty blocks. Eventually the queue must be compacted ("garbage collection") to recover these blocks. In the MMCS these high overhead periods may interfere with the running of time-critical tasks and must be avoided. To remedy this problem a stack containing pointers to all empty blocks is utilized. When an entry is added to the queue, it is placed in the location pointed to by the top of the stack and the stack is popped. When an entry is removed from the queue, a pointer to this location is pushed onto the stack. This increases the overhead for each operation but eliminates the need for the high overhead compaction periods.

The System-Program Queue. Since the system-program queue is always searched in a linear fashion it is in the form of a simple array. Each entry of the array contains the following items:

- a) ID tag
- b) Service request flag
- c) Pointer to task control block

The system-program queue structure is shown in Figure 30.

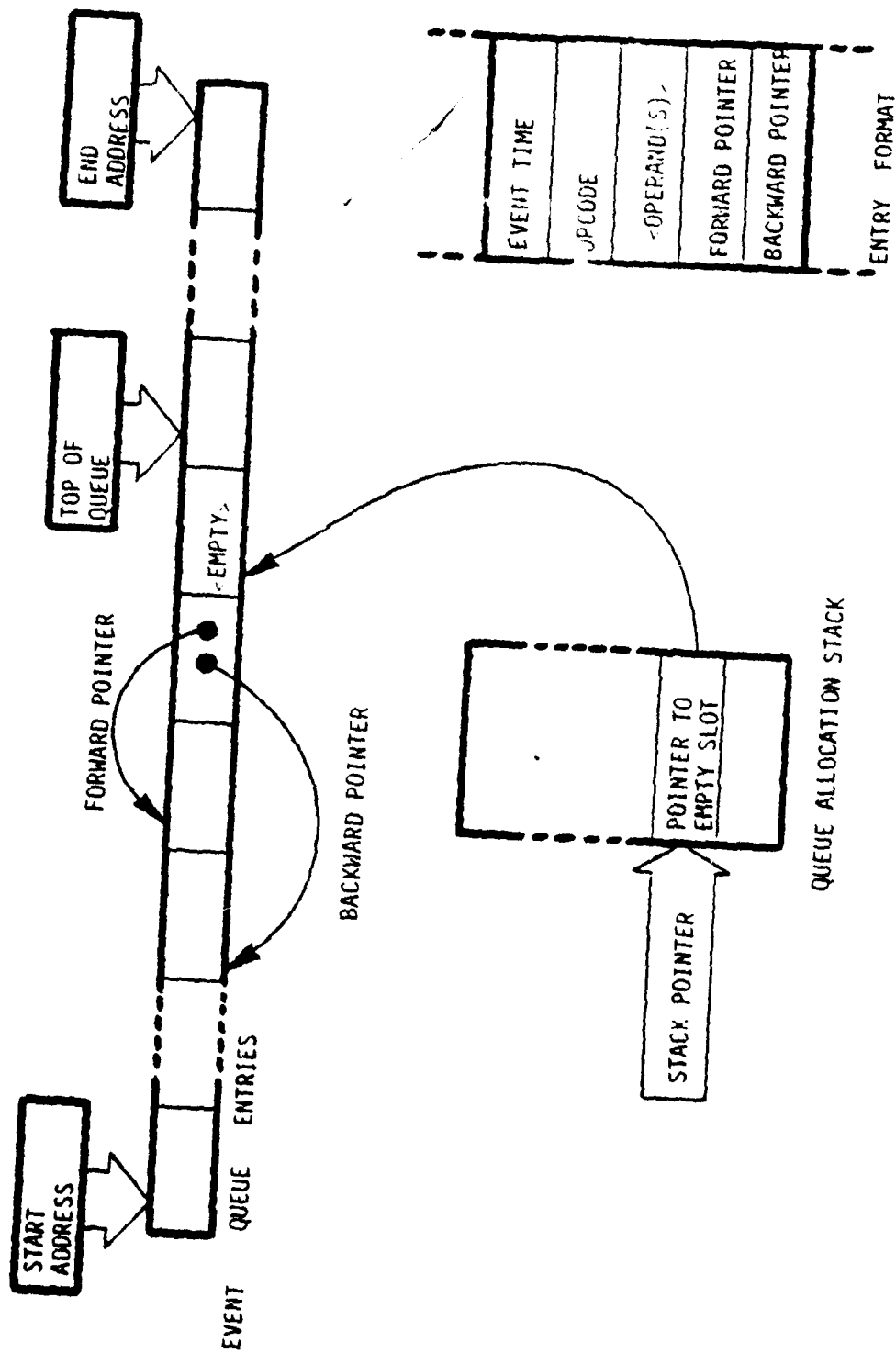


Figure 29. Event Queue Structure

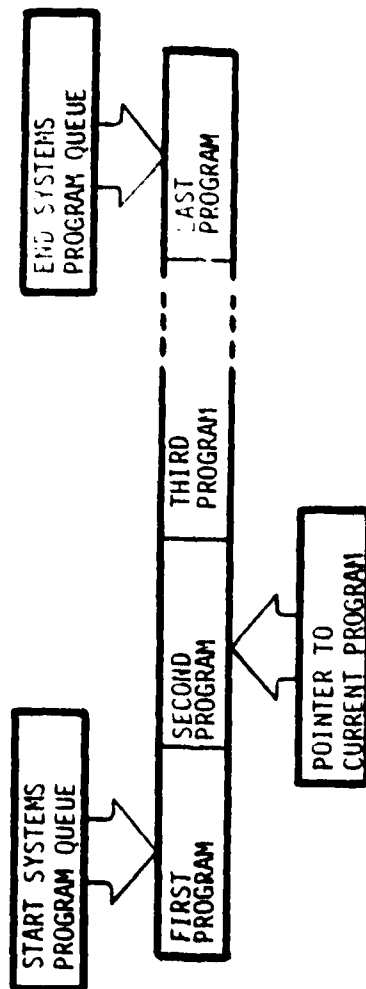


Figure 30. Structure of the Systems Program Queue

The Ready Task Queue. The RTQ is used by the application task scheduler to select the highest priority ready task. The structure of the RTQ (shown in Figure 31) is the same as the event queue except for the following differences:

- a) Entries in the RTQ are organized by priority rather than time.
- b) The RTQ contains a pointer to a specific TCB rather than an opcode and operands. This is less general than the event queue but has less overhead.

Because MMCS tasks are relatively static only a single priority is used. This decreases the overhead in searching the queue but increases the difficulty of making additions and deletions to the queue. As with the system-program queue, provision for modifications has been made by maintaining a separate queue for each priority managed by a master priority pointer queue.

Each task has a service request flag which may be located in the TCB or the RTQ. The choice is a function of the processor on which the ATM is implemented. In Figure 31 it is assumed that the service request flag is in the TCB.

ATM Implementation Requirements

The minimum hardware requirements to implement the ATM are:

- a) An ability to implement the three timekeeping functions (cycle counter, task timer, and event timer).
- b) An ability to handle the required number of interrupts; the actual number is dependent upon the complexity of the system but 24 is typical.
- c) A software interrupt to handle the supervisor calls and error traps.
- d) An ability to protect the system memory while in the user state.

The three timekeeping functions could be handled by a single hardware timer using an "alarm" mode of operation, but this would result in a large amount of overhead. A significant reduction in overhead can be obtained by utilizing vectored interrupts to eliminate the need for polling. The software interrupt (or software trap) is required to allow the use of a consistent means of entry to ATM and to ease problems of memory protection. The software interrupt also gives a controllable entry into the system area for user programs. If these features are not required then the software interrupt is not required. The fourth requirement, an ability to protect system memory, can be implemented externally if necessary.

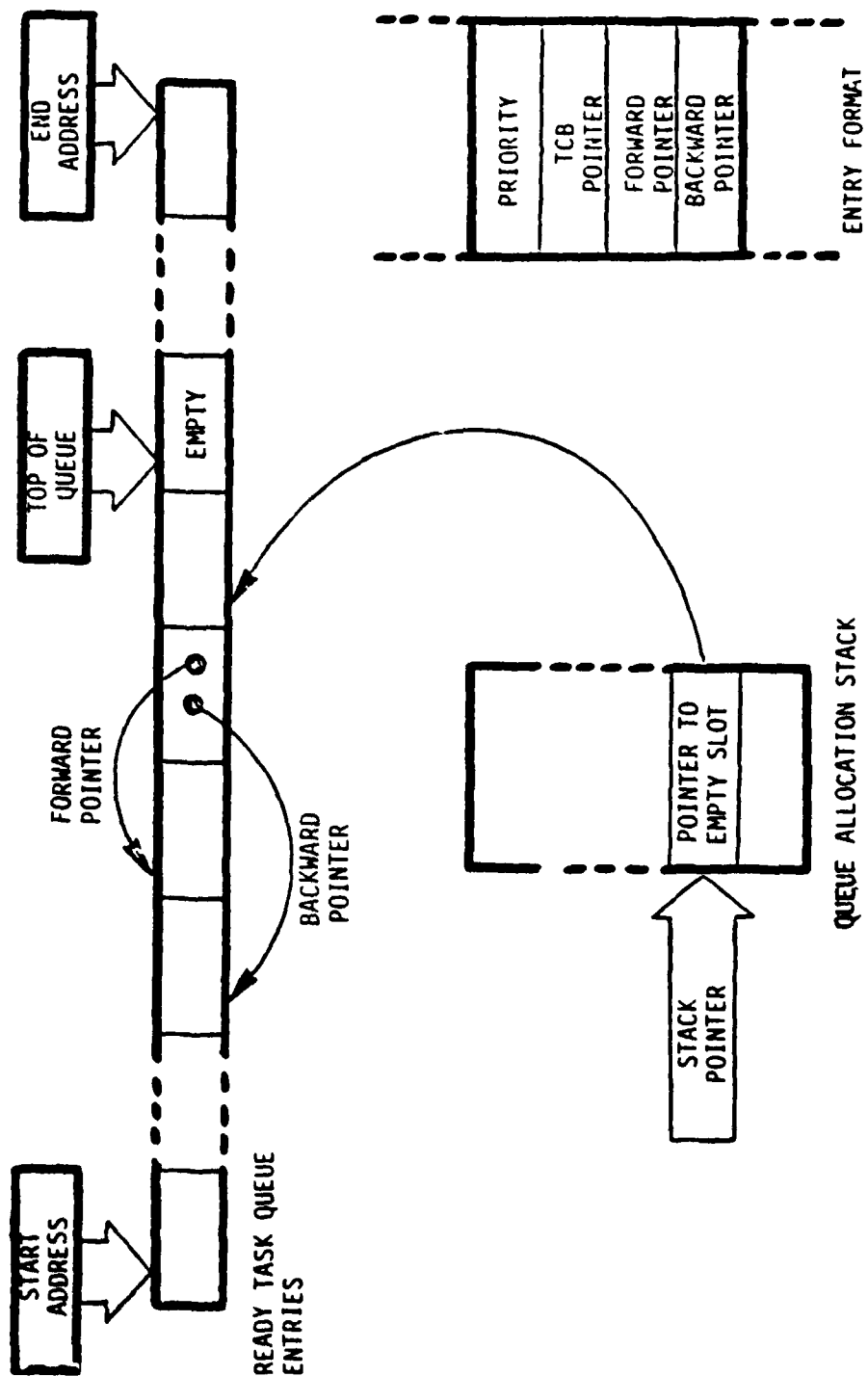


Figure 31. Ready Task Queue Structure

A major enhancement to the minimal hardware requirements would be to implement a number of functions directly in microcode. Because microcode working registers do not have to be saved across instruction boundaries, ATM could be used without having to save the status of the current user program. Most of the interrupt service functions could also be implemented in microcode. An alternative would be to use a processor which has multiple sets of working registers and flags and assign one set to ATM.

ATM Supervisor Calls

The SVC requires the use of a "software interrupt" instruction to gain entry to the SVC handler portion of the ATM. The SVC itself follows the software interrupt instruction and consists of an opcode byte followed by operand bytes as required. The format of the opcode byte is shown in Figure 32. The two most significant bits, b7 and b6, are used for special functions leaving six bits for the actual opcode. This allows up to 64 different codes. However, the opcode 63 (all 1's) is reserved as escape, allowing for unlimited expansion. Use of the escape value indicates that the following byte is to be used as the opcode.

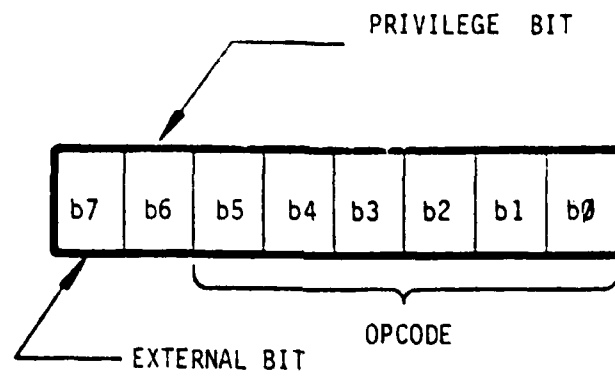
The most significant bit, b7, is defined to be the external bit. If b7=1 then the SVC refers to a task in a processor other than the one where the SVC itself is located. Where the external bit is set the opcode is followed by a logical unit number giving the address of the processor containing the task or parameter referred to by the SVC. Use of the external SVC requires cooperation between the processors involved and communication through shared memory. This would be set up at the time the tasks were loaded.

Bit b6 is used to denote a privileged SVC which has access to resources normally denied application tasks. This would normally indicate a SVC issued by a systems program.

The ATM has SVC's for the following function:

- a) Task management
- b) Flag management
- c) Interrupt control
- d) Task control
- e) Time management
- f) Error handling
- g) Resource allocation
- h) I/O and message services
- i) Event control
- j) Memory management

Additional information on the types of SVC's supported is given in Appendix S.



SVC OPCODE BYTE

Figure 32. SVC Opcode Byte

THE CONTROL PROGRAM

At the beginning of Section III it is stated that the control algorithm must perform the functions of both a real-time executive and a system control program. The real-time aspects of the control algorithm are embedded in the MMCS hardware, firmware, and software. These features are essential to the management of concurrent tasks of any type. The second function of the control algorithm, that of the system control program is application dependent. The control program must, therefore, vary from one application to another. The structure of the control algorithm is such that these variable features are indistinguishable from the fixed (or embedded) features at the application level.

The requirements for the control algorithm, as listed in Addendum E and paragraph 3.1 of the MMCC Algorithm Investigation specification, are as follows:

- a) Schedule the various microcomputers
- b) Establish the proper sequence of task executions
- c) Coordinate data transfers between common memory and the various microcomputers
- d) Coordinate the input/output functions.

The common characteristic of all of these requirements is the scheduling of events at specific times. The main job of the control program then, is to insure that these events occur as required. Examination of the four requirements reveals that the activities to be scheduled occur at all levels of the MMCS. This implies that design of the control program requires that the MMCS events be classified as to both type and scheduling mechanism. In addition, since the variable features of the control algorithm implemented by the control program must be indistinguishable from the embedded features, the control program must be integrated with the virtual machine previously described. Finally, the operation and structure of the control program must be detailed. Each of these items is discussed in this section.

Extension of the Virtual Machine Concept

The virtual machine concept was used in defining relationships between the microcomputer modules in order to remove as many hardware dependent details as possible. The same concept may be extended to cover the relationship of the MMCS and the application. This extension of the virtual machine is implemented by the control program. This is illustrated in Figure 33.

The System State/Frame Period. The frame period defined in Section I (also referred to as a system state in the MMCS specifications) is determined by the characteristics of the system simulated on the MMCS. This frame period is a basic state of the virtual machine and is implemented by the application programs. To implement a system state or frame period, the MMCS uses a number

THE TWO MAJOR COMPONENTS OF THE
CONTROL ALGORITHM:

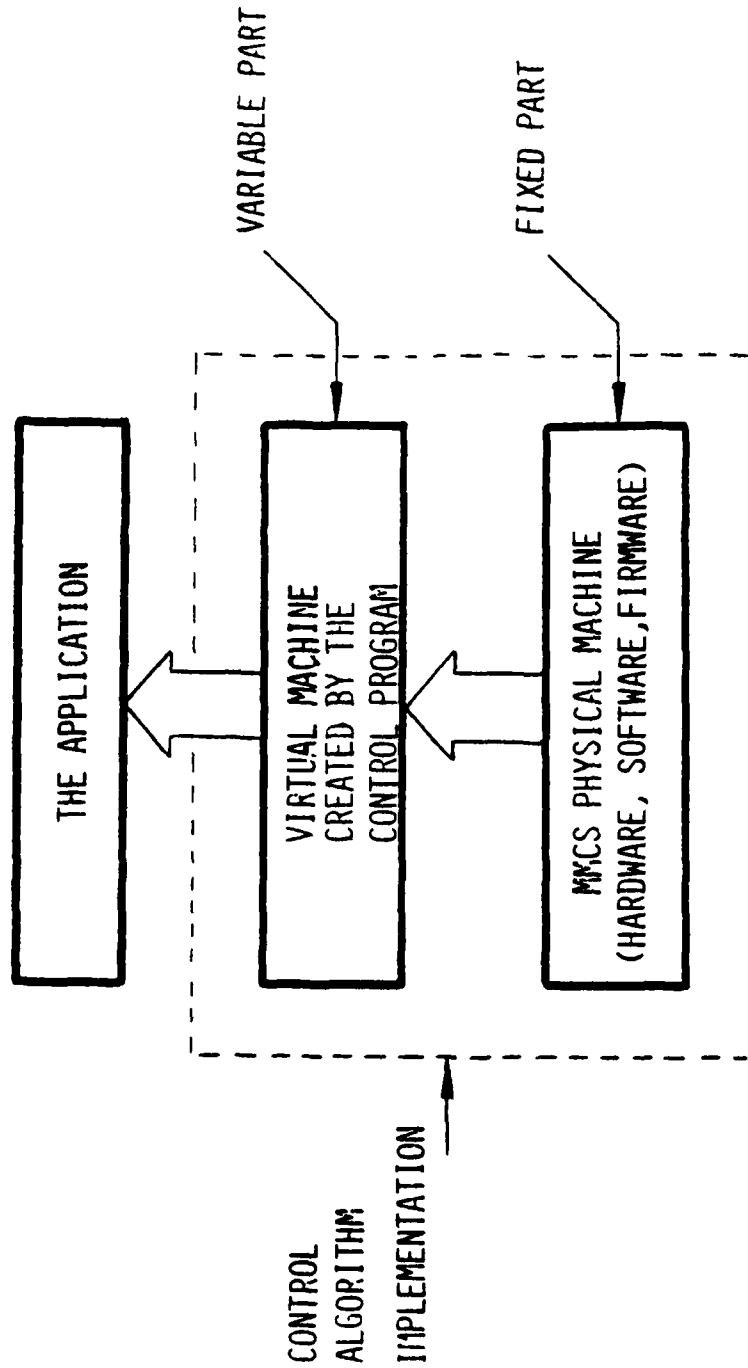


Figure 33. Extension of the Virtual Machine Concept to the Control Algorithm

of read/write cycles to the shared memory. The shared memory read/write cycles are the states of the physical machine (i.e., MMCS) on which the virtual machine is implemented. The number of cycles required for one frame is a function of the partitioning of the tasks and of the effectiveness of the control program. Efficient passing of parameters to and from shared memory can reduce the number of times a given parameter must be moved, thus increasing the availability of the shared memory bus.

Because the applications (and therefore the virtual machine) are scheduled synchronously, each frame period must have the same number of cycles or the cycle period must be variable. If the frame period has a fixed number of cycles then the number used must be the maximum required for any frame. This is not the most efficient use of the bus since it increases the number of arbitration cycles. However, since the cycle is the basic state, it is very important in scheduling the passing of system parameters and this activity would be complicated by a variable length cycle. The solution is to set some upper limit on the number of cycles per frame period and to let the cycle length be multiples of the period thus formed. All cycles are then initiated on well-defined boundaries, but a given processor only initiates cycles as required and lengthens cycles in fixed increments.

The Control Processor. The four requirements from the MMCS specifications given earlier are listed as control processor tasks. The control processor may actually initiate these activities itself or it may delegate the activity to one of the application or I/O processors. In either case, the task is scheduled by the control processor, even though it may be performed locally. The control processor then exists as both a physical and a virtual machine. The physical machine implements the virtual machine by distributing the scheduling activities to other processor modules. This distribution of the control activity decreases the traffic on the shared memory bus and makes the MMCS more responsive; an even greater advantage is that the application programs are less dependent on specific MMCS characteristics.

Classification and Scheduling of System Events

The events scheduled by the control processor are classified as global, distributed, and application events. Their relative positions in the MMCS structure are shown in Figure 34. The types of events are the same for all classes although certain types may be more likely to appear in one particular classification. Events may be scheduled on a synchronous, asynchronous, or exceptional basis. Events in any classification may be scheduled on any basis, although the distribution is not the same for all classifications.

Classification of Events. Global events are handled by the portion of the control program that resides in the control processor. Some typical global events are: down-loading information or programs to processor modules, executing system level dynamics, and handling errors at the system level. The events

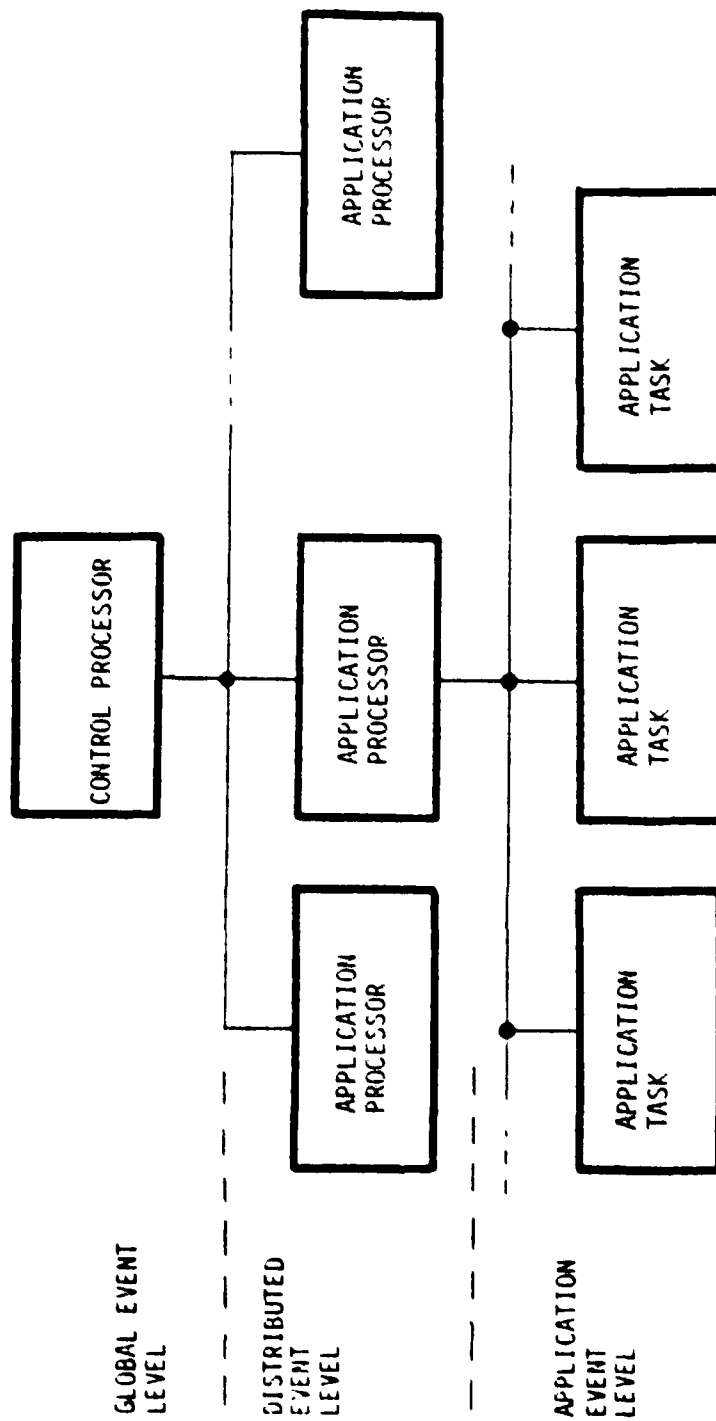


Figure 34. Placement of Event Classes Within MICS Structure

handled by the control processor are primarily exceptional events because routine activities are distributed among the other processor modules.

Distributed events are handled by processor modules at the system (ATM) level. Typical distributed events include the passing of parameters to and from shared memory and the initiating or terminating of tasks. This is the level where the bulk of the control program resides. Most of the tasks of the virtual control processor are handled at this level.

Application events are those which may be initiated by an application task or a device under the control of that task. Application events allow functions, such as I/O, to be handled outside of the control program. These events may make a disciplined use of system resources by use of supervisor calls.

Scheduling of Events. Synchronous scheduling of events takes less system overhead than asynchronous or exceptional scheduling. It is used for all events which occur on a periodic predictable basis. Multiple, related synchronous events must be sequenced so that precedence is preserved.

Asynchronous scheduling is used for tightly coupled or nonperiodic events. When the producer (sender) of a parameter is asynchronous but the consumer (receiver) is synchronous, very little extra overhead is involved. For more tightly coupled situations where control or handshaking information must be passed, asynchronous scheduling can require substantial overhead. The application tasks are not allowed to directly modify each other so the communication between tasks is by the supervisor calls provided for this purpose. There are two types of supervisor calls which are useful in the asynchronous scheduling of events. One involves the use of discretely, which are flags that are modified or tested by the application tasks through the use of supervisor calls. A given discrete is assigned to a single task and only this task can modify (set or clear) the discrete. However, all tasks may test the discrete by using a supervisor call to return the value of the discrete or by using a supervisor call to suspend or terminate a task based on a specified value for the discrete.

The second type of supervisor call used in asynchronous scheduling is a semaphore. There are two possible operations on a semaphore v , SIGNAL (v) AND WAIT (v), as defined in Section I. The variable v represents a memory location or locations. It is incremented by one when a task executes a SIGNAL supervisor and decremented by one (unless it is already zero) when a task executes a WAIT supervisor call. If it is already zero the task executing the WAIT will be suspended until the semaphore is incremented by a SIGNAL operation. This is done by placing an opcode and operands on the event queue, causing the semaphore to be checked after a specified time interval. If the semaphore is zero the process is repeated; otherwise the task is flagged as ready and runs when it has the highest priority. The intervals at which the system checks the

semaphore are selectable. The semaphore is not owned by any one task and is the basic building block for more complicated operations. It also is a higher overhead activity than the discrete.

Both the discrete and semaphore supervisor calls can be used with tasks in separate processors, in which case they are maintained in shared memory. Operations on discretess or semaphores in shared memory request the bus by use of the priority request lines so their latency time is low. They do, however, add to system overhead as well as increase traffic on the shared memory bus.

Exceptional scheduling is used for events which are outside of normal operation. An example of exceptional scheduling is the use of error traps or system alarms.

Control Program Operation and Structure

The control program is the software portion of the control algorithm. It represents the application-dependent (and user-accessible) portion of the control algorithm. To the greatest extent possible the operational portion of the control program (the synchronous and asynchronous events) is distributed among the various microcomputer modules. This reduces the traffic on the common buses and vastly increases the parallelism of the system. An implication of this is that the virtual control processor is more powerful than the physical control processor. The control program in each processor has three major parts:

- a) The cycle program
- b) The distributed control program (supervisor calls)
- c) The exceptional event handlers

The Cycle Program. Each processor of the MMCS has a cycle program. The cycle program is run as the highest priority system program and is the heart of the control program. The structure of the cycle program is shown in Figure 35. The cycle program is initiated by an interrupt from the cycle counter. For each cycle there is a pointer to a queue of activities for that cycle, and the chief activity is to handle all the synchronous, distributed events that are to occur. This involves flagging the required application tasks to run and setting up the passing of parameters to and from shared memory. The queue for a cycle contains pointers to the task control blocks of the application tasks, making it simple to flag them to run. Each task has a time limit but the tasks usually terminate themselves with a supervisor call after they have produced the required result. Within a given cycle the precedence of the tasks is handled by priority. In some instances a chain of asynchronous tasks is initiated by the cycle program.

The passing of parameters is done by providing pointers to the proper lists to a system program that handles this activity. Bus requests are made by the cycle program and the bus grants are connected to assignable interrupts

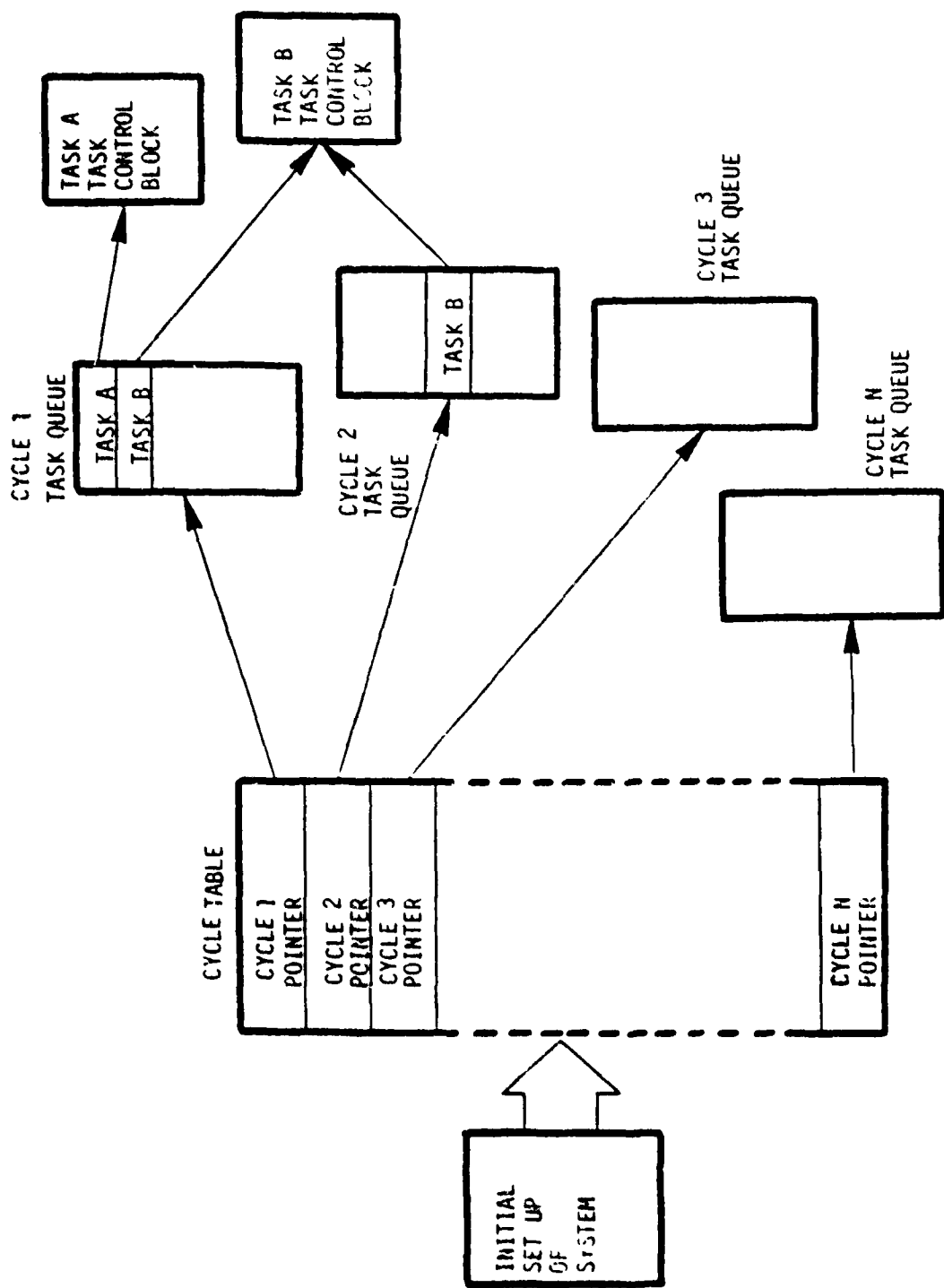


Figure 35. Organization of the Cycle Program

which cause the parameter program to run. Each time a bus grant is received the program is initiated and the ATM is entered. The parameter program uses the pointers supplied by the cycle program to make the actual transfer to or from shared memory.

The cycle program may also perform other tasks. The last task performed is one that loads the cycle counter with the number of cycles until the next interrupt. The design of the cycle program is such that speed is optimized at the expense of memory. It is also structured so that run-time changes are possible.

The Distributed Control Program. A significant portion of the control program is distributed among the application tasks in the form of supervisor calls, whose use for the asynchronous scheduling of events and for the termination of most synchronous tasks has been discussed previously. In addition, some tasks may handle their own scheduling by suspending themselves for an interval of time. This allows a task to be scheduled asynchronously with very little system overhead. These supervisor calls constitute a substantial part of the control program.

The Exceptional Event Handlers. The ATM contains handlers for exceptional events such as errors. If required, these handlers may initiate a user task or communicate with the control processor.

Diagnostics. Each microcomputer module, including the control processor, has a diagnostic program which checks the system status to the greatest extent possible. This routine is made to run as a background task by giving it the lowest possible priority, but unlimited time.

System Initialization

To initialize the multiple microcomputer system a system manager uses the console connected to the control processor and an initialization program which runs on the control processor as a system program. The bulk storage device, typically a disk, is handled by the control processor and holds the programs for all processors, including the control processor. The control processor has a small disk-boot program in ROM which is used to load its programs from the disk. These programs are then used to load object code and operating parameters into the other modules. The control bus is used to down-load the code, leaving the shared memory bus free for other activities. As each processor is loaded it is placed in the WAIT state. When all processors have been loaded the operator then starts the system when desired. The initialization program includes cold-start diagnostic routines also.

Control Processor Tasks

The major tasks handled by the control processor are:

- a) Initialization
- b) Dynamic assignment of tasks
- c) System-level diagnostics
- d) System-level error handling
- e) Timebase management
- f) Keeping all system records
- g) Handling operator interaction.

The major requirements of the control algorithm are included in tasks (a) and (b). Due to the distributed nature of the control algorithm, most of the run-time decisions take place in the application processors. Because the control bus is used for initialization, it is possible for it to change the assignment of an application processor module during execution. This could be used to re-configure the system or to keep a processor failure from interrupting a run.

Although each processor has its own diagnostic and error handling routines, these activities are handled at the system level by the control processor. The routines for this are application programs on the control processor. These programs are expected to be continually updated as a given application matures and to occupy a large portion of the resources of the control processor.

To reduce system overhead the main timebase is handled by the control processor. Once each frame it places in shared memory the present system time and the frame number. If possible, these values are assigned to the distributed cache to reduce the bus load. The system time is derived from the system clock available to every processor. This allows incremental times within the frame to be handled locally.

The other major control processor functions are record keeping and handling operator interaction with the MMCS.

SECTION IV

MICROCOMPUTER MODULE DESIGN AND ANALYSIS

The microcomputer module(s) for the MMCS consists of a processor module and a memory module with their associated buses. These modules are designed in accordance with the capability requirements of paragraph 3.3 of NTEC Specification N74-105. Specifically, the modules are fundamentally identical, interface to static hardware buses, are microprogrammable, have a word length of 32 bits, and have components available from at least two sources. The processor module meets the performance requirements of paragraphs 3.3 and 3.4 by emulating the instruction set of a DEC VAX-11/780.¹⁶ Because the modules are microprogrammable, other native instruction repertoires may be implemented to meet specific trainer system requirements.

The easiest and most economical way to achieve this emulation is by the use of bit-slice processor parts. These parts allow design flexibility in both hardware and instruction sets, while maintaining high performance. At present there are six families of bit-slice microprogrammable processor sets. These are the 2900-series from Advanced Micro Devices, the Macrologic-series from Fairchild, the 3000-series from Intel, the 6701-series from Monolithic Memories, the 400-series from Texas Instruments, and the 10800-series from Motorola. All of these families are manufactured using Schottky-TTL technology, except for the 10800-series which utilizes ECL.

Each of the bit-slice processor families can be used to emulate a VAX-11/780, but some of the families have disadvantages relative to the rest. The 400-series from Texas Instruments is not second-sourced, rendering it unacceptable for a design which must have a long service lifetime. The Fairchild Macrologic-series provides the lowest performance of the six families and is best suited for controller and processor applications requiring limited capabilities. The Intel 3000-series has only a 2-bit wide slice, so twice as many processor elements are needed compared to the other families. Also, because its register file has only a single port, more instructions and longer execution times are required for some tasks, making it a poor choice for computationally intensive applications. This family has good I/O capabilities, however, and is the best choice for data manipulation. The 6701 bit-slice processor from MMI and the 2901 device from AMD are similar in capabilities and features and are well-suited to high-performance processor designs. However, AMD has recently offered a refined version, the AMD 2903, with additional arithmetic and logic instructions including multiply and divide. This device has the most features of any bit-slice processor and is hence the best for computationally intensive applications. In addition, the AMD 2910 microprogram controller has the greatest degree of integration and the most features, in terms of branch

16. VAX 11/780 Architecture Handbook, Volume 1, Digital Equipment Corporation, Maynard, MA, 1977, pp. 5-3 through 5-33.

and subroutine capabilities, of any controller available. The last family to be considered, the MC 10800-series from Motorola, has the highest throughput due to its ECL construction, but requires voltage-level conversions to be compatible with logic support devices which are mostly TTL. The device provides no multiply or divide capability and has no internal working registers. Based on these considerations, the AMD 2900-series has been chosen for the implementation of the microcomputer processor module. The 2900-series is a well-supported and well-documented family of devices that offers the following advantages:

- a) It has the widest variety of functions available as individual chips which reduces design complexity.
- b) It is directly compatible with TTL components, requiring no logic-level conversions as do ECL components.
- c) The 2900-series components have the largest degree of integration, so that a design based on these would have the fewest parts.

ARCHITECTURE OF THE MICROCOMPUTER MODULE

The microcomputer (without memory), shown in Figure 36, implements a portion of the VAX instruction set and all of its addressing modes. The design centers around three separate buses. The address bus accesses both local and shared memory. Data is transmitted via a separate data bus, again with provision for a path to either local or shared memory. The control signals received or sent by the processor are handled by a separate control bus.

The major paths of data flow in the processor are through a set of input registers and buffers. The information present can come from either the address bus or the data bus. The information is then passed to 16 working registers. The VAX general purpose registers are internal to the 2903's. From any of these registers, the data is sent to the arithmetic logic unit (ALU) where data manipulation is performed. The output of the ALU is available to either the address bus (via an address register) or the data bus (via a data register). The ALU output can also be directed to a program counter and a status register.

Instructions are fetched from the data bus 4 bytes at a time and held in 4 opcode registers. One byte, denoting the macro-instruction, is decoded as a starting address in the control store. Addressing the control store is the function of the microprogram sequencer. The output of the control store is latched in a microinstruction register to provide microcontrol signals.

Data Paths to the ALU

A more detailed description of the processor modules is now presented, beginning with a discussion of data paths to the ALU, as shown in Figure 37.

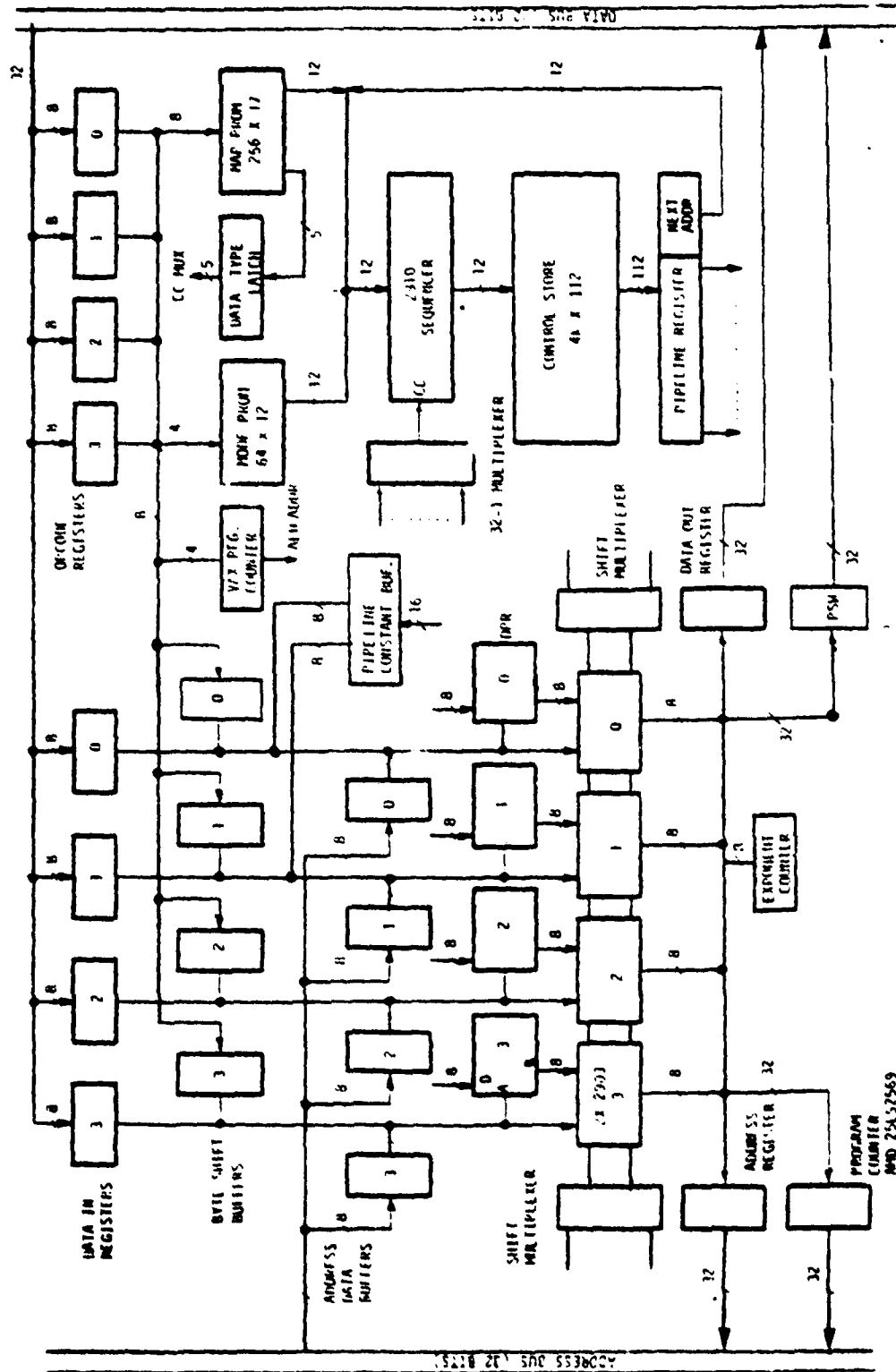


Figure 36. Hardware Block Diagram of a Processor Module

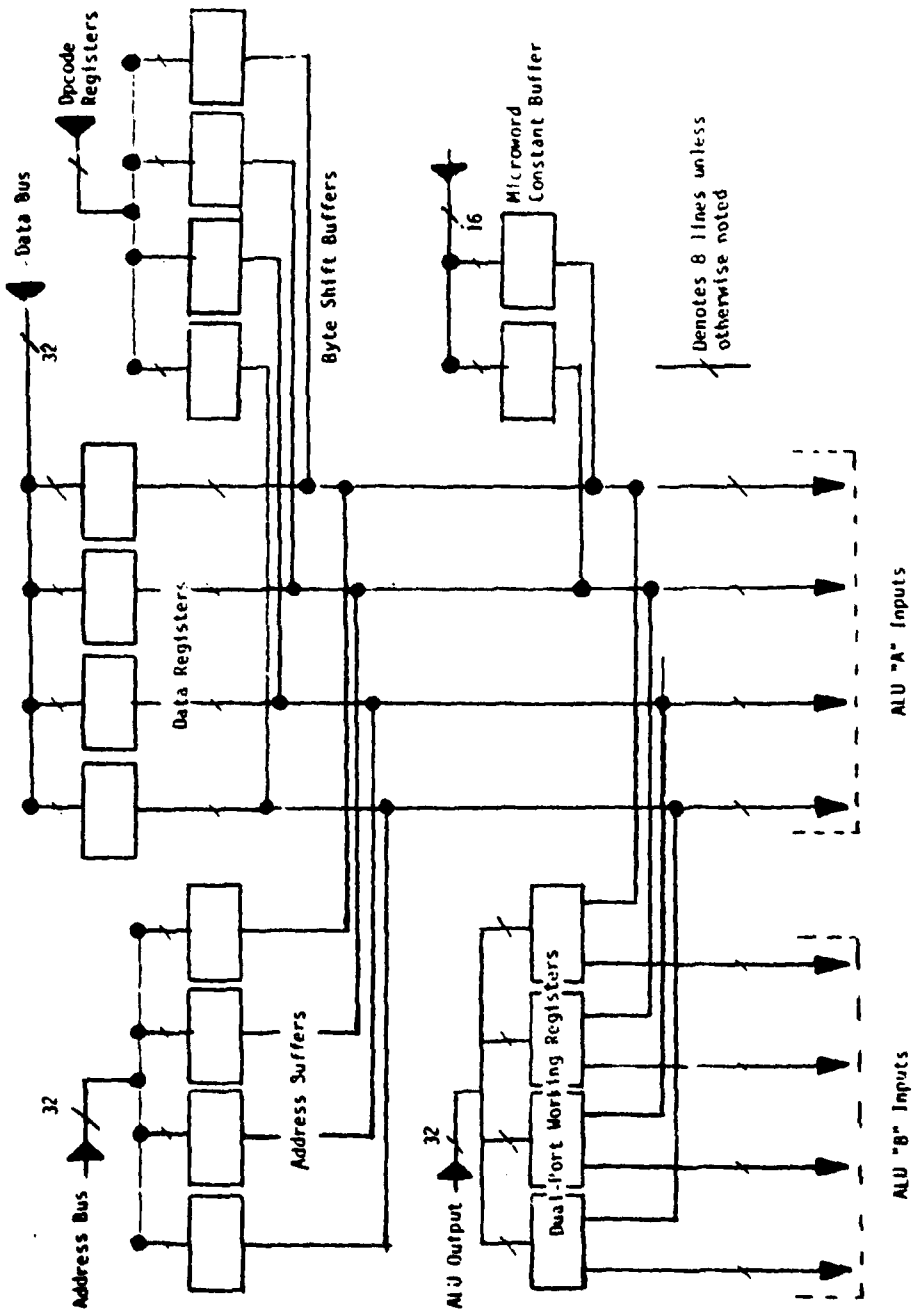


Figure 37. Data Paths to the ALU

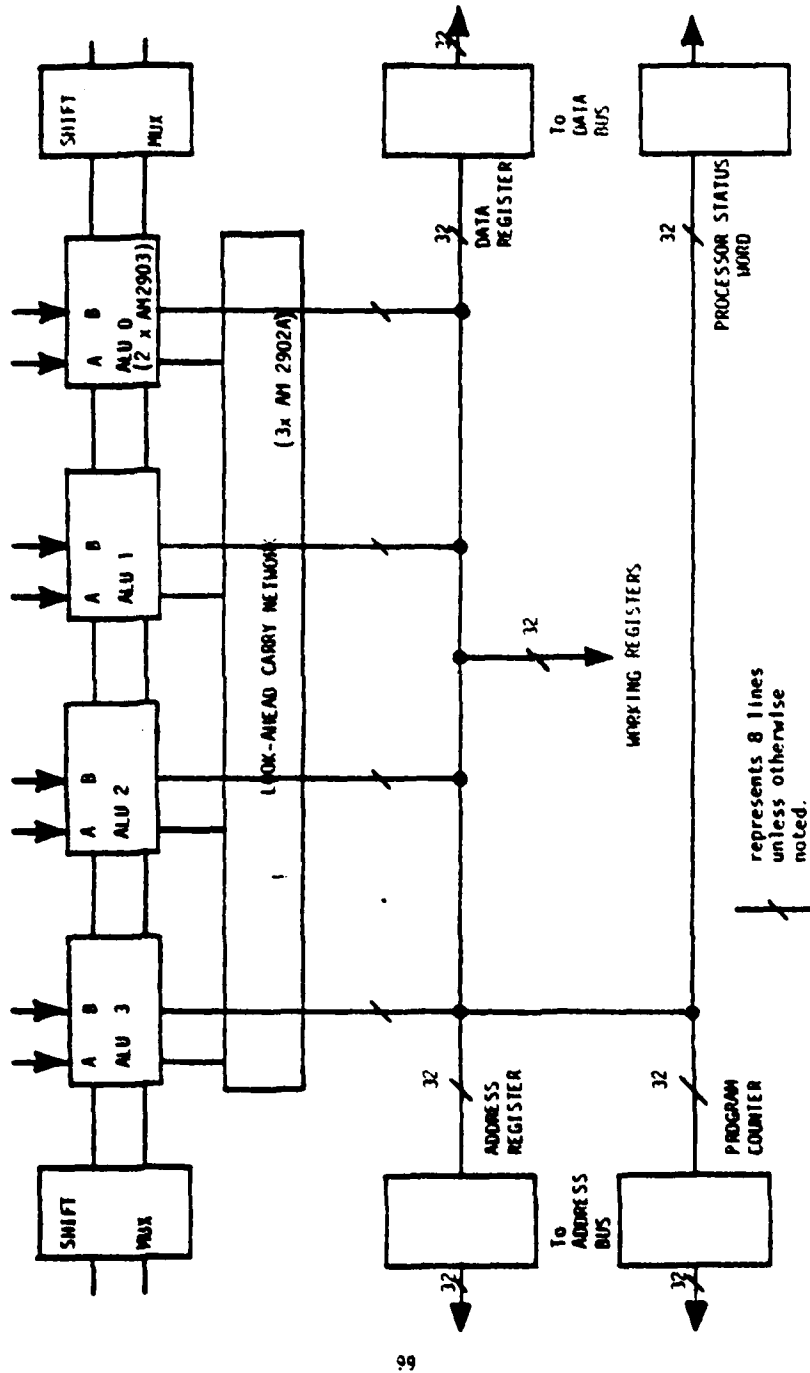


Figure 38. Block Diagram of Arithmetic-Logic Unit



Figure 39. Block Diagram of Instruction Decoding Logic

Data present on the 32-bit data bus can be latched into a data register, which is organized as four one-byte registers. The register output is available in any combination of bytes to the "A" data input of the 2903 microprocessor slices.

The information present in the 32-bit address register is available to the "A" input of the 2903 via the address bus and 4 one-byte address buffers. The output of the 4 opcode registers is also buffered for routing to the "A" input port.

The processor working register consists of a group of 16-word two-port RAM's. One output port is available to the "A" input of the 2903; the other to the "B" input. The data input of the working register group is connected to the output of the ALU section so that processed data may be returned to the registers for further use. Finally, data may be sent to the "A" input of the ALU from the microword constant buffer to allow the data present in the microword to be transferred to the ALU.

ALU Design

The arithmetic-logic unit of the processor is based on the 2903 bipolar microprocessor slice (see Figure 38). This device contains 16 internal registers. There are three data paths within the 2903: "A" and "B" inputs which are fed directly to the internal ALU section of the device and a "Y" input which is connected to the input port of the internal registers. The internal ALU output is available to the internal registers on the same path as the "Y" input.

The ALU output is available to 5 different register sets: an address register and a program counter, both connected to the address bus; a data register and processor status register, both connected to the data bus; and finally, the 16 working registers. The program counter is made external to the ALU chips to provide increased processor speed; it consists of eight 25LS2569 four-bit up/down counters with three-state outputs.

For the 32-bit configuration chosen, the ALU consists of eight 4-bit slices grouped as four bytes. A look-ahead-carry network consisting of 3 2902A high-speed look-ahead carry generators is also present. Shift multiplexers are present at the most and least significant bytes of the ALU for arithmetic and logical computations.

Instruction Decoding Logic

Instructions are fetched from memory 4 bytes at a time and latched into a group of opcode registers. Because the VAX instructions are of variable length, instructions are decoded one byte at a time. Figure 39 contains a block diagram of the instruction decoding logic.

The VAX 11/780 has a set of 244 instructions; therefore one byte is required to encode a given instruction type. The MAP PROM has 256 addresses and maps the opcode byte into a starting address in the control store for the instruction under consideration. The MAP PROM has 16 output bits; 12 bits are used for addressing the $4k(=2^{12})$ of control store and the remaining bits are used to specify one of the following possible data types:

- a) byte
- b) word (2 bytes)
- c) longword (4 bytes)
- d) quad word (8 bytes)
- e) floating point/not floating point

The data type information is held in a data-type latch.

The MODE PROM is used in the implementation of the addressing modes of the VAX instruction set. In the design discussed here, each addressing mode is executed as a subroutine in the control store and a separate routine is provided for the source or destination operand specifier of each mode in most cases. However, four of the 16 general register addressing modes and one of the 8 program counter addressing modes are valid for source operands only. This restriction limits the total number of valid addressing modes to 43. The MODE PROM has 64 addressable locations, some of which will be used to implement addressing mode faults. The output of the MODE PROM is 12 bits wide to provide 4K addressing capability.

A separate counter is used to hold the address of the VAX register used for register operations. This counter is incremented for quad word operation because two sequential registers are required for the storage of quad data words. For example, if a quad word contained in R0 and R1 is to be transferred to R2 and R3, the transfer proceeds as R2←R0 and R3←R1. The register counter is incremented after the first transfer to point to the second of the two registers for the second transfer.

Microprogram Sequencer and Control Store

Input address information to the microprogram sequencer, a 2910 Microprogram Controller is available from three sources; starting addresses from the MAP and MODE PROM'S and next address information contained in the microinstruction register. Condition code information is provided to the sequencer via a 32-1 multiplexer. There are 4K microwords in the control store, each of which is 112 bits wide. The output of the control store is latched in a register. From this register the individual control bits are available to the ALU, registers, buffers, and other control points. The address of the next microinstruction is held in the microword, which is available to the ALU via a 16-bit constant buffer. This sequencer is detailed in Figure 40.

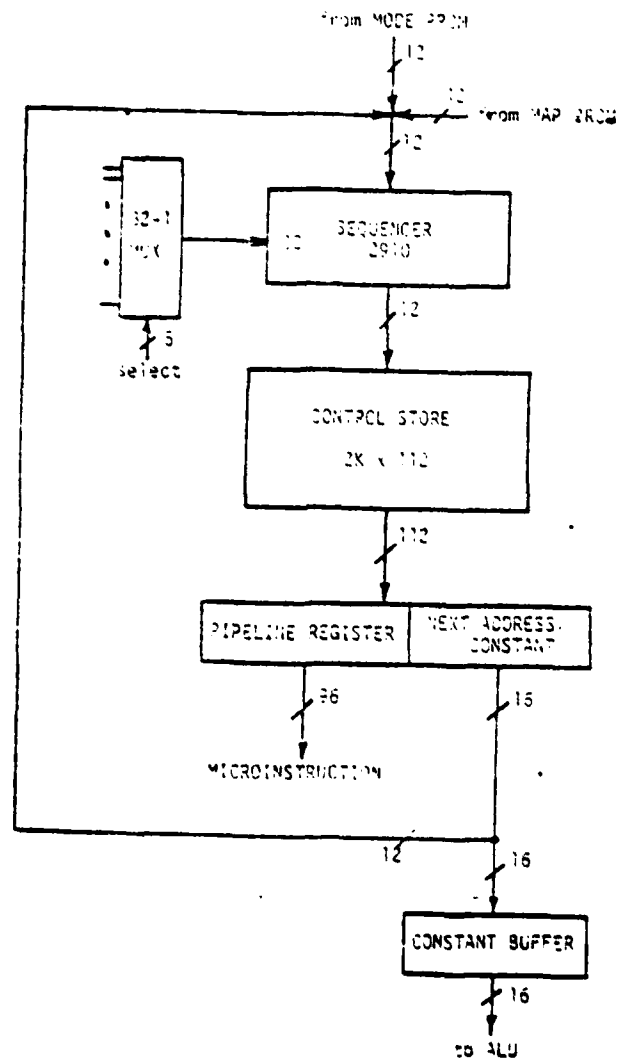


Figure 40. Microprogram Sequencer and Control Store

PROCESSOR MODULE FIRMWARE

The processor module is designed to emulate a DEC VAX-11/780 computer by implementing a subset of the instructions and all of the addressing modes of a VAX for the same data types. The instructions, listed in Table 3, include the 18 instructions required by Addendum B to Specification N74-105 and also indicated in NAVTRAEQUIPCEN IH-261. The instructions are microcoded for storage in a PROM control store. In addition, single and double-precision floating-point instructions and the real-time task manager (ATM) are microcoded for inclusion in the control store. All microcode is written using structured programming and modular design techniques.

Microprogram Structure

The microprogram structure is based on a hierarchical, top-down design which consists of a short main program and many subroutines as shown in Figure 41. An example of the main program is shown in Figure 42. All instructions within the same block in this figure are executed concurrently. After initialization, the main program executes a FETCH which places the first four bytes of opcode into the opcode registers and the program counter is incremented by one. (The program counter then points to the next byte.) The output of the first opcode register is enabled using combinational logic based on the two least significant bits of the program counter. This output information is latched in the address register of the MAP prom.

The next group of instructions attempts a FETCH. However, the control of FETCHES is done with an external hardware counter which keeps track of the location of the current active opcode register. When all four bytes of the opcode register have been used, the FETCH is permitted. This structure is necessary because the main program with its nested subroutines cannot easily keep a record of the current, active opcode register. With the attempted FETCH, the program counter is incremented and the output of the next opcode register is enabled. Concurrent with these operations, a subroutine jump to the microprogram address specified by the MAP PROM is executed. (The MAP PROM decodes its one byte of input information into a starting address for the macro-instruction type specified by that byte.)

The micro-subroutine for a particular macro-instruction consists primarily of a source operand subroutine and destination operand subroutine. An example micro-subroutine for a macro-instruction is shown in Figure 43. Following the source and destination subroutines, a FETCH is attempted, the program counter incremented, the next opcode register output enabled, and a jump to the MAP address executed. The VAX addressing mode 05, register direct, is shown in Figures 44 and 45 for source and destination operands. The source subroutine moves the contents of the VAX register to an internal source register, RSRC, within the ALU array. A FETCH is attempted and the VAX register counter

TABLE 3. VAX 11/780 INSTRUCTIONS MICROCODED

| <u>MNEMONIC</u> | <u>INSTRUCTION</u> | <u>MNEMONIC</u> | <u>INSTRUCTION</u> |
|-----------------|-------------------------------------|-----------------|---|
| ACBB | Add compare and branch byte | BICW2 | Bit clear word 2 operand |
| ACBD | Add compare and branch double | BICW3 | Bit clear word 3 operand |
| ACBL | Add compare and branch long | BISB3 | Bit set byte 3 operand |
| ACBW | Add compare and branch word | BISL2 | Bit set long 2 operand |
| ADDB2 | Add byte 2 operand | BISL3 | Bit set long 3 operand |
| ADDB3 | Add byte 3 operand | BISW2 | Bit set word 2 operand |
| ADDD2 | Add double 2 operand | BISW3 | Bit set word 3 operand |
| ADDD3 | Add double 3 operand | BITB | Bit test byte |
| ADDF2 | Add floating 2 operand | BITL | Bit test long |
| ADDF3 | Add floating 3 operand | BITW | Bit test word |
| ADDL2 | Add long 2 operand | BLBC | Branch on low bit clear |
| ADDL3 | Add long 3 operand | BLBS | Branch on low bit set |
| ADUW2 | Add word 2 operand | BLEQ | Branch on less or equal |
| ADW3 | Add word 3 operand | BLEQU | Branch on less or equal unsigned |
| ADWC | Add with carry | BLSS | Branch on less |
| AOBLEQ | Add one and branch on less or equal | RLSSU | Branch on less unsigned |
| AOBLS | Add one and branch on less | BNEQ | Branch on not equal |
| ASHL | Arithmetic shift long | BNEQU | Branch on not equal unsigned |
| BBC | Branch on bit clear | BPT | Break point fault |
| BBCS | Branch on bit clear and set | BRB | Branch with byte displacement |
| BBS | Branch on bit set | BRW | Branch with word displacement |
| BBSC | Branch on bit set and clear | BSBB | Branch to subroutine with byte displacement |
| BCC | Branch on carry clear | BSBW | Branch to subroutine with word displacement |
| BCS | Branch on carry set | CASEB | Case byte |
| BEOL | Branch on equal | CASEL | Case long |
| BEOLU | Branch on equal unsigned | CASEW | Case word |
| BGEQ | Branch on greater or equal | CLRB | Clear byte |
| BGEQU | Branch on greater or equal unsigned | CLRD | Clear double |
| BGTR | Branch on greater | CLRF | Clear float |
| BGTRU | Branch on greater unsigned | CLRL | Clear long |
| BICB2 | Bit clear byte 2 operand | CLRQ | Clear quad |
| BICB3 | Bit clear byte 3 operand | CLRW | Clear word |
| BICL2 | Bit clear long 2 operand | CMPB | Compare byte |
| BICL3 | Bit clear long 3 operand | CMPL | Compare long |

AD-A080 735

SOUTH CAROLINA UNIV COLUMBIA DEPT OF ELECTRICAL AND --ETC F/6 9/2
MULTIPLE MICROCOMPUTER CONTROL ALGORITHM.(U)
SEP 79 R O PETTUS, R D BONNELL, M N HUMNS N61336-78-C-0157

UNCLASSIFIED

NAVTRAEQUIPC-78-C-0157-1 NL

2 of 3

ALL
AD-A080 735

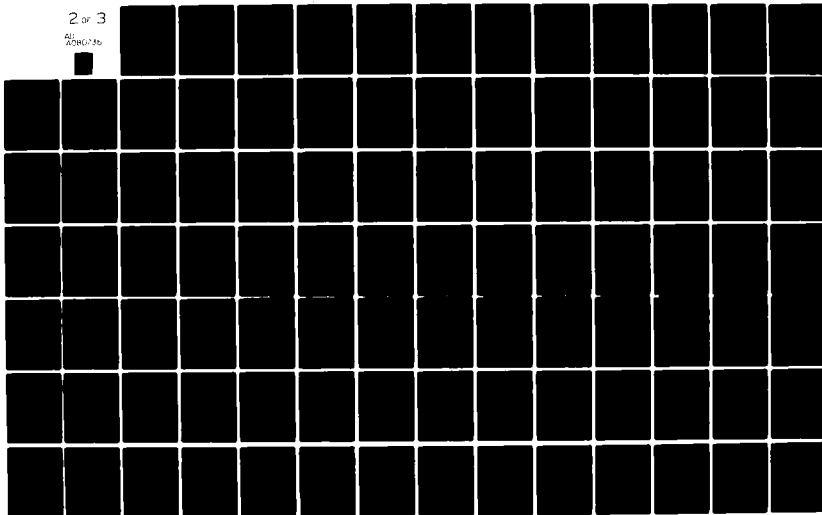


TABLE 3. VAX 11/780 INSTRUCTIONS MICROCODED (cont.)

| <u>MNEMONIC</u> | <u>INSTRUCTION</u> | <u>MNEMONIC</u> | <u>INSTRUCTION</u> |
|-----------------|---------------------------------|-----------------|---|
| CMPW | Compare word | MULB3 | Multiply byte 3 operand |
| CVTBL | Convert byte to long | MULF2 | Multiply floating 2 operand |
| CVTBW | Convert byte to word | MULF3 | Multiply floating 3 operand |
| CVTWL | Convert word to long | MULL2 | Multiply long 2 operand |
| DECB | Decrement byte | MULL3 | Multiply long 3 operand |
| DECL | Decrement long | MULW2 | Multiply word 2 operand |
| DECH | Decrement word | MULW3 | Multiply word 3 operand |
| DIVF2 | Divide floating 2 operand | POPR | Pop registers |
| DIVF3 | Divide floating 3 operand | PUSHL | Push long |
| DIVL2 | Divide long 2 operand | PUSHR | Push registers |
| DIVL3 | Divide long 3 operand | ROTL | Rotate long |
| INCB | Increment byte | RSB | Return from subroutine |
| INCL | Increment long | SOBGEQ | Subtract one and branch on greater or equal |
| INCH | Increment word | SOBGTR | Subtract one and branch on greater |
| JMP | Jump | SUBB2 | Subtract byte 2 operand |
| JSB | Jump to subroutine | SUBB3 | Subtract byte 3 operand |
| MATCHC | Match characters | SUBD2 | Subtract double 2 operand |
| MCOHB | Move complemented byte | SUBD3 | Subtract double 3 operand |
| MCOML | Move complemented long | SUBF2 | Subtract floating 2 operand |
| MCOMW | Move complemented word | SUBF3 | Subtract floating 3 operand |
| MNEGB | Move negated byte | SUBL2 | Subtract long 2 operand |
| MNEGL | Move negated long | SUBL3 | Subtract long 3 operand |
| MNEGW | Move negated word | SUBW2 | Subtract word 2 operand |
| MOVB | Move byte | SUBW3 | Subtract word 3 operand |
| MOVD | Move double | TSTB | Test byte |
| MOVF | Move float | TSTL | Test long |
| MOVL | Move long | TSTW | Test word |
| MOVQ | Move quad | XORB2 | Exclusive OR byte 2 operand |
| MOVW | Move word | XORB3 | Exclusive OR byte 3 operand |
| MOVZBL | Move zero-extended byte to long | XORL2 | Exclusive OR long 2 operand |
| MOVZBW | Move zero-extended byte to word | XORL3 | Exclusive OR long 3 operand |
| MOVZWL | Move zero-extended word to long | XORW2 | Exclusive OR word 2 operand |
| MULB2 | Multiply byte 2 operand | XORW3 | Exclusive OR word 3 operand |

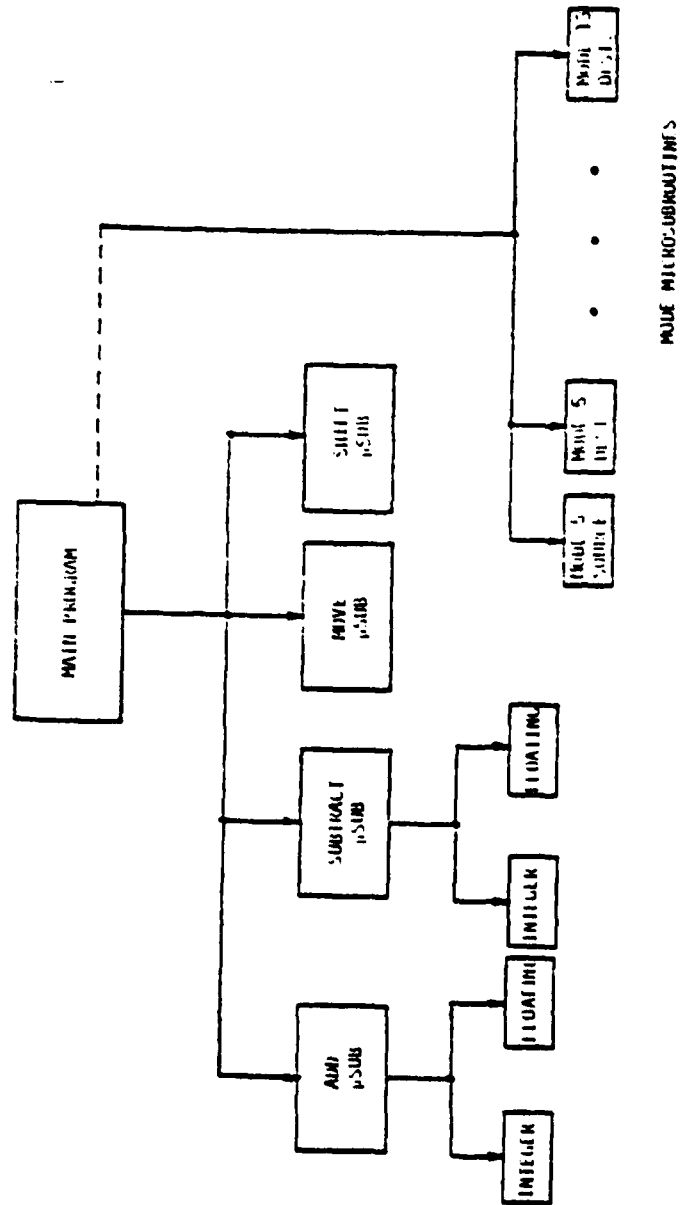


Figure 41. Functional Block Diagram of the Microprogram Structure

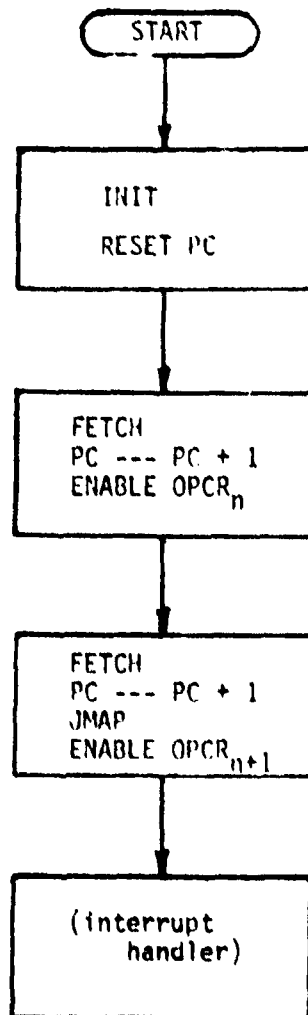


Figure 42. Flowchart for the Main Microprogram

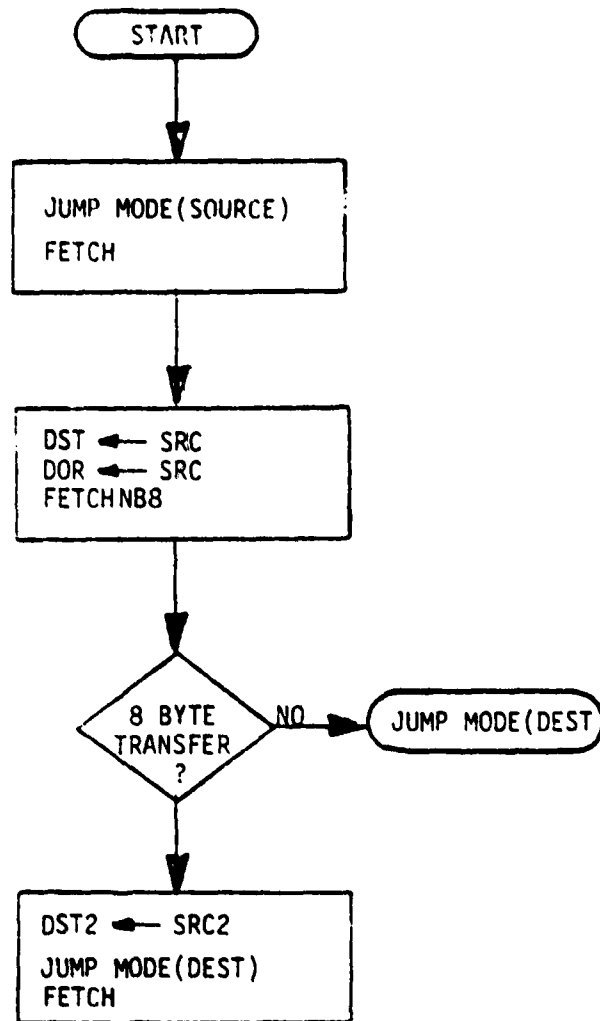
MOV_ MICROSUBROUTINE

Figure 43. Flow Chart for the Microsubroutine for the MOV_ Instruction.

MODE 5 (REGISTER DIRECT) MICROSUBROUTINE

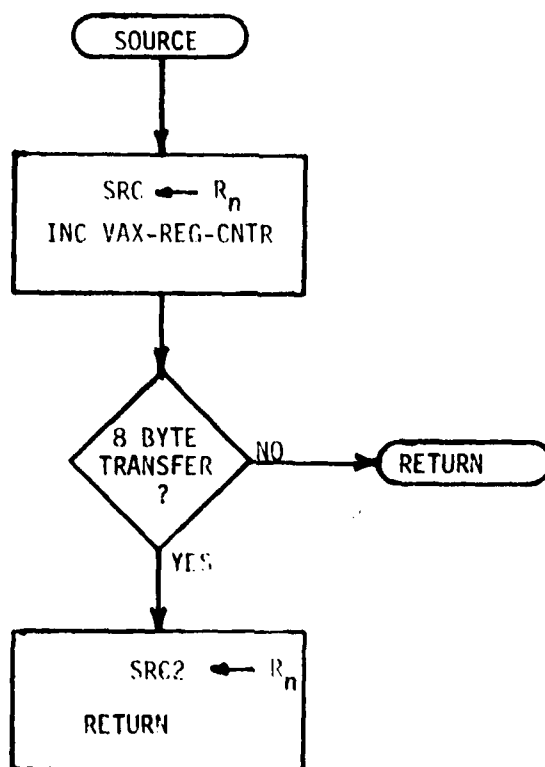


Figure 44. Microsubroutine for the Register-direct Source Operand.

MODE 5 (REGISTER DIRECT) MICROSUBROUTINE

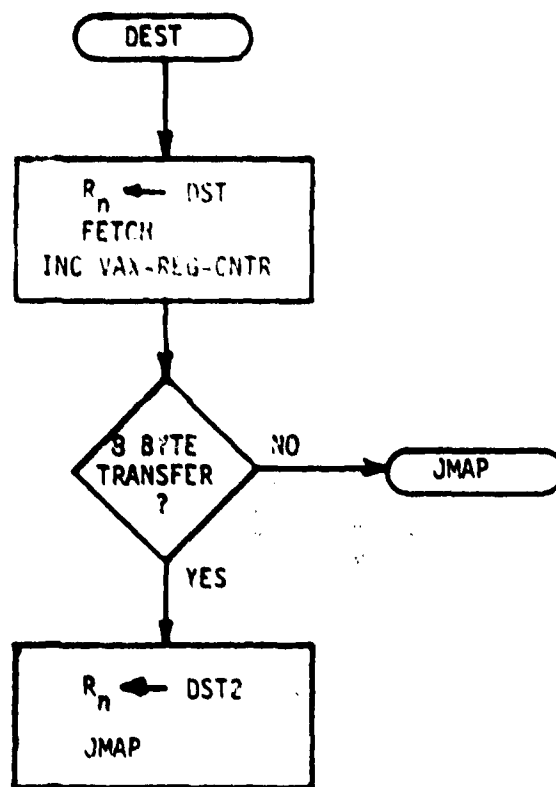


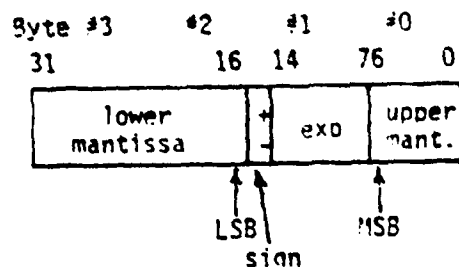
Figure 45. Microsubroutine for Register-direct-mode Destinations

(Figure 39) is incremented during the same microcycle. The counter is incremented to point to the next VAX register so that in the event of an 8-byte (two register) transfer the register counter is already pointing to the correct register. This saves a microcycle. If an 8-byte transfer is not to be executed, control returns to the macro-instruction subroutine.

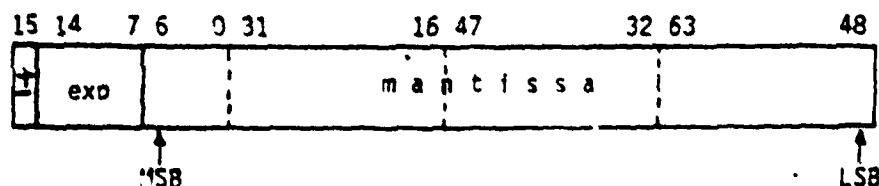
Control of all internal processor functions is governed by a 112-bit microinstruction word, whose format is shown in Figures 46, 47, and 48. This microinstruction word is also described by the microprogram assembler definition table contained in Appendix C. This definition table is used by the microprogram assembler during its assembly phase to create the binary microprogram to be loaded into the control store. Appendix D shows the assembly phase output for the instruction subset and all addressing microsubroutines.

Floating-Point Logic and Microcode

The VAX-11/780 instruction set includes both single-precision (32 bits) and double-precision (64 bits) floating-point operations. These operations are addition, subtraction, multiplication, division, data movement, data conversion, and data testing. The floating-point operands for these instructions have the following formats:



Single-Precision Format



Double-Precision Format

The exponents, which must be in the range from -127 to +127, are stored in excess-128 code. (This is obtained by adding 128 to the exponent and storing the 8-bit magnitude of the result.) An excess-128 exponent of zero is used to indicate a true numerical value of zero.

NAVTRAEQUIPCEN 78-C-0157-1

| | |
|-----|------------------------------|
| 111 | UNDEFINED |
| 106 | LOAD EXPONENT COUNTER |
| 105 | REGISTER LOAD |
| 100 | |
| 99 | |
| 98 | CONDITIONAL FETCH |
| 9 | MEMORY CONTROL |
| 91 | |
| 90 | |
| 89 | UPDATE SIGN F/F #1 |
| 88 | UPDATE SIGN F/F #0 |
| 87 | INCREMENT VAX-REG-CNTR |
| 84 | DATA/ADDRESS MODE |
| 83 | ADDRESS AND DATA BUS CONTROL |
| 82 | |
| 81 | |
| 80 | EXP. COUNTER DIRECTION |
| 79 | ENABLE EXP. COUNTER |
| 78 | SET INDEX MODE F/F |
| 76 | CLEAR INDEX MODE F/F |
| 75 | MISC. CONTROL OF 56-71 |
| 72 | SEQUENCER D INPUT SEL. |
| | SEQUENCER INSTRUCTION |
| | |

Figure 46. Microword Format for Bits 111 - 72.

| | |
|----|---|
| 71 | NEXT ADDRESS OR CONSTANT FIELD |
| 56 | |
| 55 | |
| 54 | |
| 53 | 2910 REG. LOAD ENABLE |
| 52 | SOURCE / DESTINATION |
| 48 | CC POLARITY |
| 47 | CONDITION CODE SELECT |
| 45 | |
| 44 | |
| 39 | |
| 38 | SOURCE SELECT FOR 2903 "A" INPUT |
| 37 | DUAL-PORT WORKING REGISTER "A" ADDRESS |
| 32 | DUAL-PORT WORKING REGISTER "B" ADDRESS |

Figure 47. Microword Format for Bits 71 - 32.

| | |
|----|---|
| 31 | ALU FUNCTION |
| 28 | |
| 27 | ALU DESTINATION OR SPECIAL FUNCTION |
| 22 | |
| 21 | DUAL-PORT REGISTER WRITE ENABLE |
| 17 | |
| 16 | CARRY-IN SELECT |
| 14 | |
| 13 | FLAG CHANGE CONTROLS |
| 10 | |
| 9 | FLAG INPUT SOURCE |
| 5 | |
| 4 | SHIFT MUX INPUT SELECT |
| 1 | |
| 0 | UNDEFINED |

Figure 48. Microword Format for Bits 31 - 0.

Non-zero floating-point numbers have a unique representation when they are normalized according to the relationship $1/2 \leq f < 1$, where f is the fractional part or mantissa of the number. This relationship causes the most-significant bit (MSB) of f always to be a one. Hence this MSB does not have to be stored, although it must be restored before any floating-point operations are performed. The resultant representations have a precision of one part in 2^{23} for single-precision numbers and one part in 2^{55} for double-precision numbers. An accuracy of $\pm 1/2$ least-significant bit (LSB) is maintained during all floating-point operations by using two extra guard bits on the right of the mantissa and an overflow bit on the left of the mantissa.

The above floating-point conventions and requirements are met for single-precision, floating-point addition by the following algorithm:

- a) Rotate operand #1 left one bit and store in working register; test for zero exponent
- b) Rotate operand #2 left one bit and store in working register; test for zero exponent
- c) Subtract bytes #1 from working registers and store result in 8-bit exponent counter ($op\#1 - op\#2$)
- d) Rotate working registers one bit to the right while shifting a one into the MSB of bytes #0
- e) Compute: Result sign = $(op\#2 \text{ sign}) \oplus (\text{Exp. counter MSB}) + (op\#1 \text{ sign}) \cdot (\text{Not exp. counter MSB})$
- f) (1) If exponent counter ≥ 0 , zero byte #1 of working register #2 and shift working register #2 to right while decrementing exp. counter until exp. counter = 0
(2) If exp. counter ≤ 0 , zero byte #1 of working register #1 and shift working register #1 to right while decrementing exp. counter to zero.
- g) Move exponent of unshifted working register into exp. counter and zero byte #1 of that working register
- h) Add working registers
- i) Normalize:
 - (1) If carry from byte #0 = 1, then no normalizing is needed
 - (2) If carry from byte #0 = 0, rotate result left until shifted output from byte #0 = 1; decrement exponent counter for each shift and test for zero exponent (underflow).
- j) Replace exponent from exp. counter in byte #1 of result.
- k) Rotate result right one bit, while inserting sign @ MSB of byte #1.

The above algorithm and similar ones for subtraction, multiplication, and division have been designed, analyzed, and microcoded.

ATM LOGIC AND MICROCODE

The Applications Task Manager (ATM) described in Section III will be microcoded to minimize the amount of overhead it requires to operate. It will require a minimum of additional logic on the processor module--only logic for controlling the processor status register(s) and the processor buses. The ATM microcode will handle all interrupts and all communications between processors and between tasks. This microcode will be stored in a block at the top of the control-store memory space.

MICROCODE-CONTROL-STORE MODULE DESIGN

The control store for the microcode is designed as a separate module to be connected to the processor module by means of a 124-line cable and two 64-pin connectors. The module, shown in Figure 49, contains 2K x 112-bits of high-speed bipolar PROM. This PROM is constructed from fourteen Intel 3636-1 chips, each containing 16K bits organized as 2K x 8-bits. The access time for each chip is 65 nanoseconds. The 12 input address lines (from the 2910 Sequencer on the processor module) are fully buffered to minimize the load on the 2910. The 112 output data lines do not need buffering because they are connected directly to the pipeline register located on the processor module for a fan-out of one.

The use of a separate control-store module has two advantages. First, it results in a simpler processor module and one which can be tested independently from its control store. Second, the control-store module is designed to be plug-compatible with the writable-control-store cards used in microprogramming development systems. This allows microcode and processor hardware to be completely debugged and tested in a development system before being committed to permanent form in a PROM. After the 3636-1 PROMS are programmed with corrected microcode, the control-store module can be substituted for the writable control store and the system made operational.

MEMORY MODULE DESIGN

The memory modules for both local memory and shared memory are constructed using static random-access, 4-kilobit memory chips, organized as 4K x 1-bit. Each memory module contains 4096 32-bit words, requiring 32 memory chips per module. Because the processor has a microcycle time of 200-250 nanoseconds, the memory module must have a cycle time less than 150 nanoseconds, including the delays for decoding and memory management. The requirements are satisfied by Intel 2147 4K x 1 static RAMS. (Static RAMS are chosen over dynamic RAMS because of the savings in hardware and time resulting from the elimination of refresh circuitry and refresh microcycles.) Addresses are fully decoded on each memory module and all bus lines are buffered to prevent bus loading--each module presents at most one TTL load to each bus line.

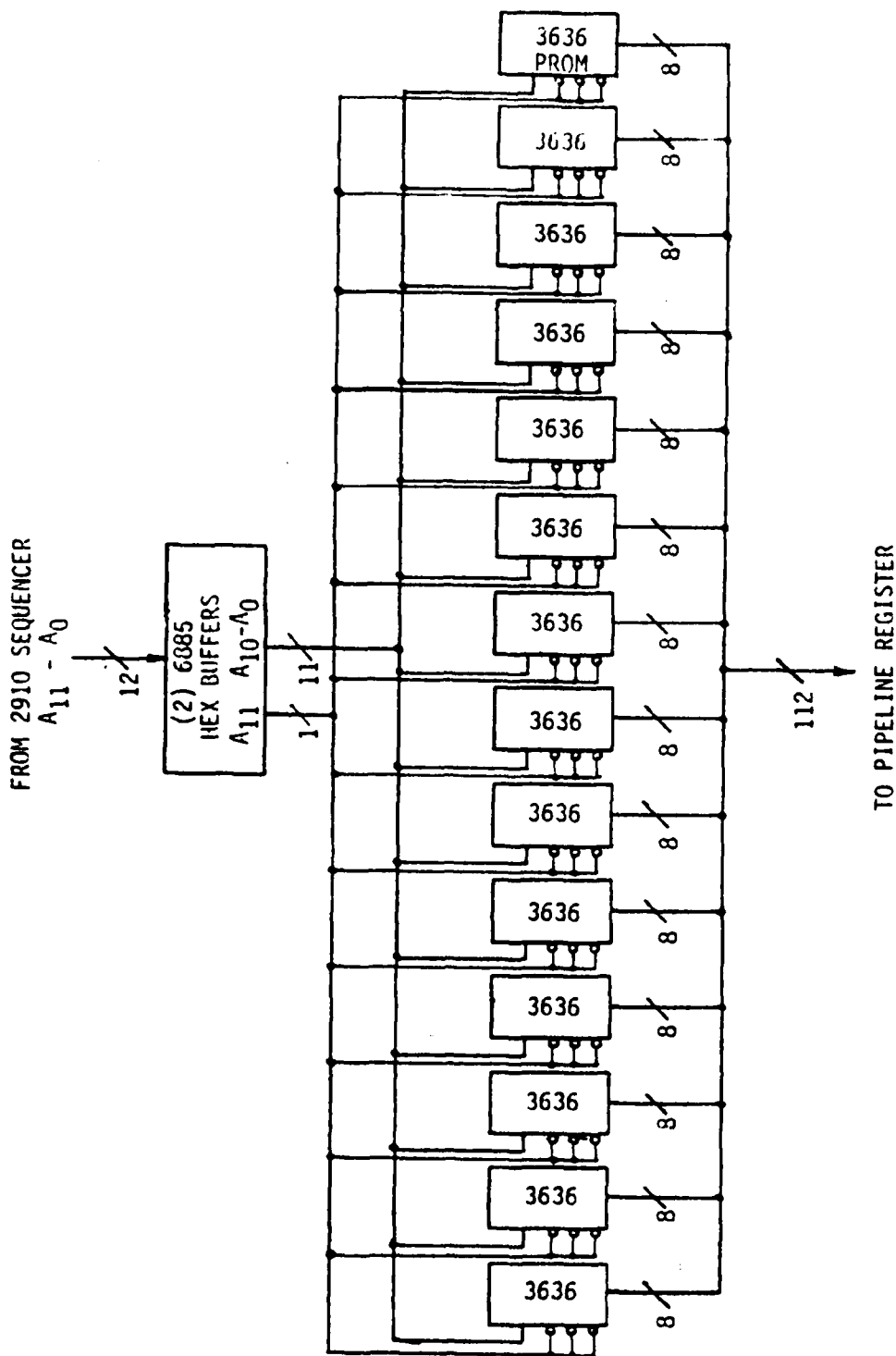


Figure 49. Microcode Control Store PROM Module.

A memory-alignment unit is required between the processor module and the memory modules because the VAX instructions being emulated allow memory accesses on any byte boundary, while the memory is organized as 32-bit words. Figures 50 through 54 show the bus interfaces to the memory-alignment unit and a complete logic design for this unit; this configuration handles nonaligned memory-read and memory-write operations. Two microcycles are required for a nonaligned read or write operation, compared to one microcycle for a memory operation involving aligned data. To avoid this execution time penalty, programmers must carefully construct their algorithms to minimize the number of nonaligned-data operations.

Figure 50 shows the interfaces between the processor module and the four major buses of the multiple-microcomputer system. All connections to these buses are through the Memory Alignment Unit. The memory-address space of the processor is divided into four parts and one part is assigned to each of the four buses. The four most-significant bits (A_{31} to A_{28}) of the 32-bit address sent out by the processor control the access to these buses.

The input and output signals for the Memory Alignment Unit are shown in Figure 51. Because nonaligned read and write operations require two microcycles, the Memory Alignment Unit must be a sequential machine with two states. A state diagram for this machine is shown in Figure 52 along with the logic circuit needed to implement a controller for the states. Also shown is the circuit for generating the signal that indicates if the address is aligned or not. These circuits control the rearrangement of the data during read and write operations, as shown in Figure 53. Figure 54 contains the circuitry for generating the write enables to each of the four bytes in a memory word.

PERFORMANCE EVALUATION

The execution time of the machine (MACRO) instruction is determined by both the number of microinstructions required per machine instruction and the microinstruction cycle time, now estimated at 200 nanoseconds. Instruction times are minimized by short microprograms and fast microcycle times. The microcycle times are determined by data path analysis in the ALU and its support circuitry.

The instructions of Addendum B of NTEC Document N75-105 have been micro-coded and their execution times are listed in Table 4. The instruction execution time for a given instruction depends on a number of factors which are described below so it is not possible to give a specific execution time for each. Rather a best-case and worst-case time is provided.

The computation of the average instruction time is based on the specified usage factors and the best case and worst case execution times for the instruc-

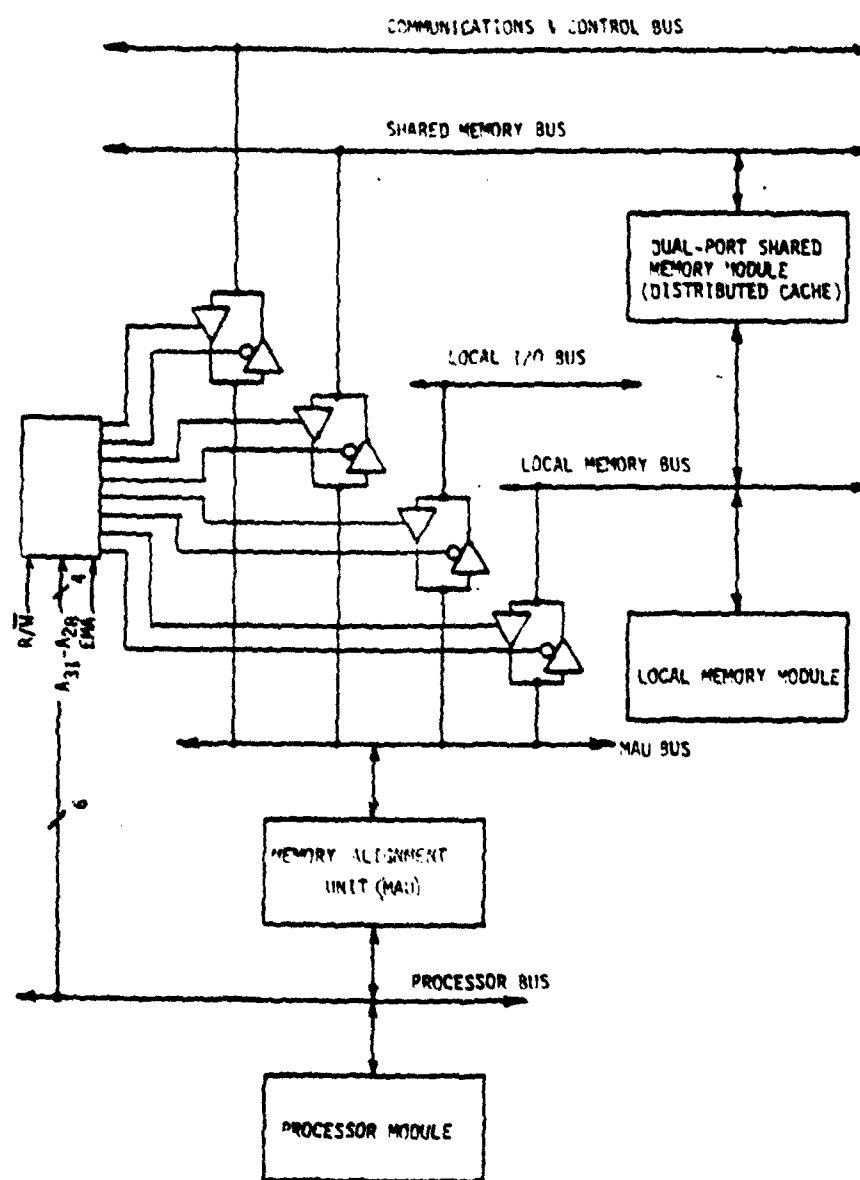


Figure 50. Microcomputer Module Block Diagram and Bus Interfaces

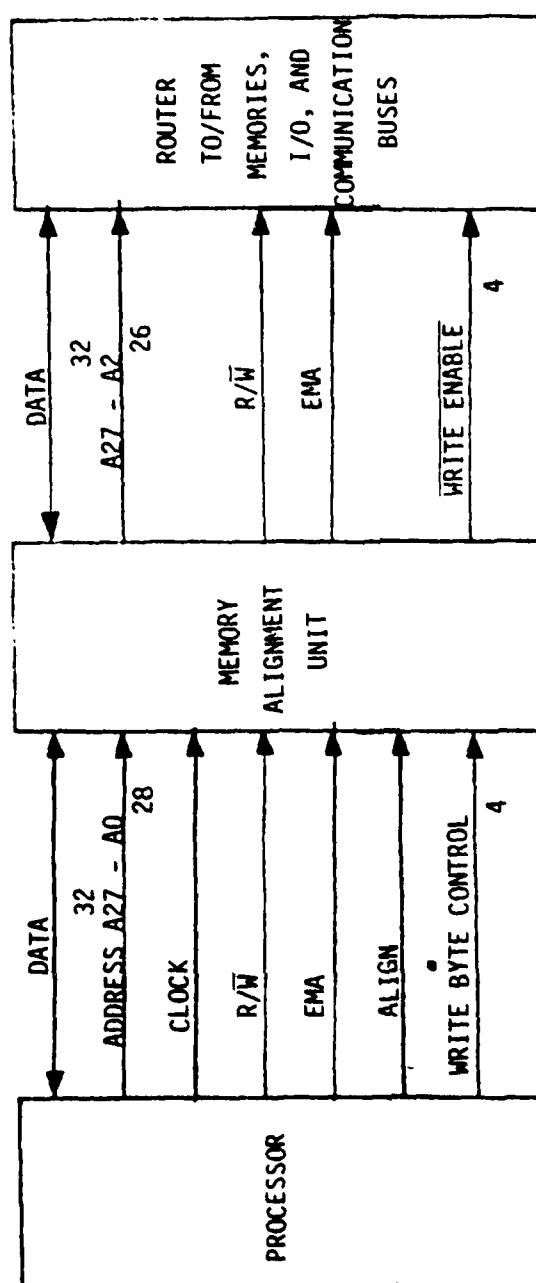
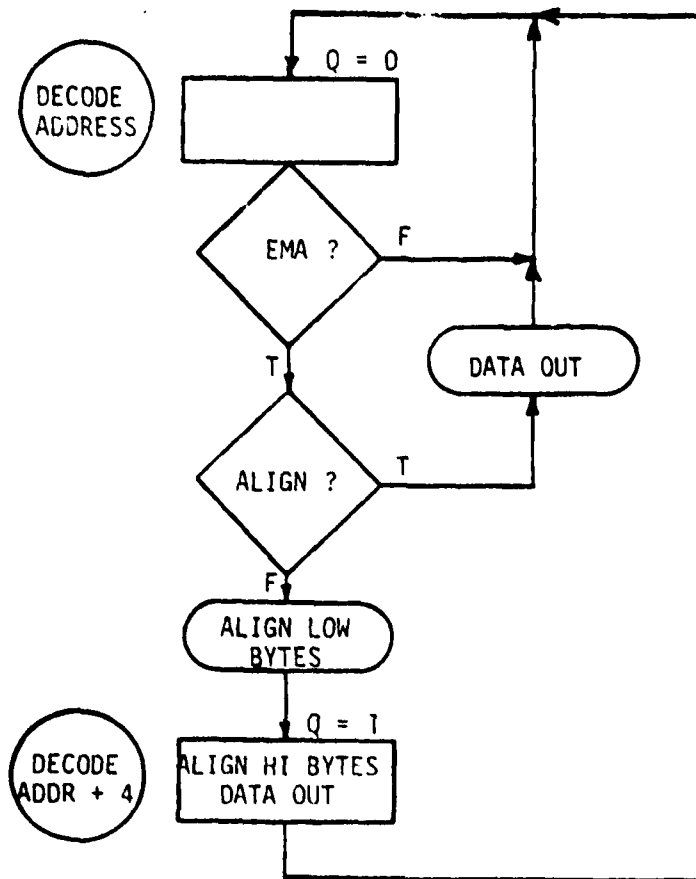


Figure 51. Input and Output Signals for the Memory-Alignment Unit.

Algorithmic State Machine Chart:



Implementation:

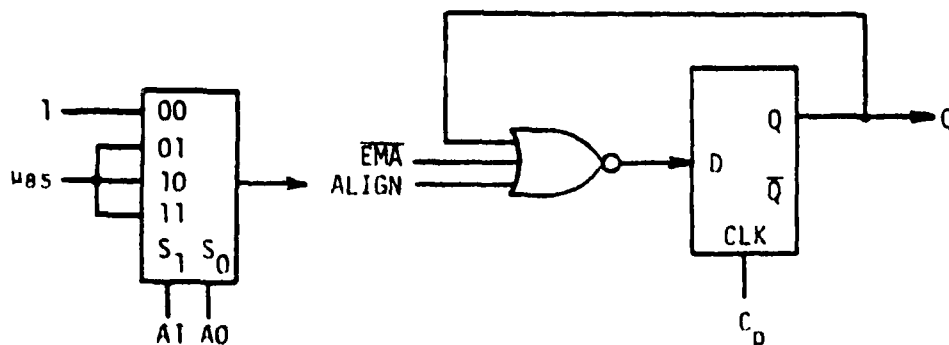


Figure 52. ASM Chart for Memory-Alignment Unit Controller

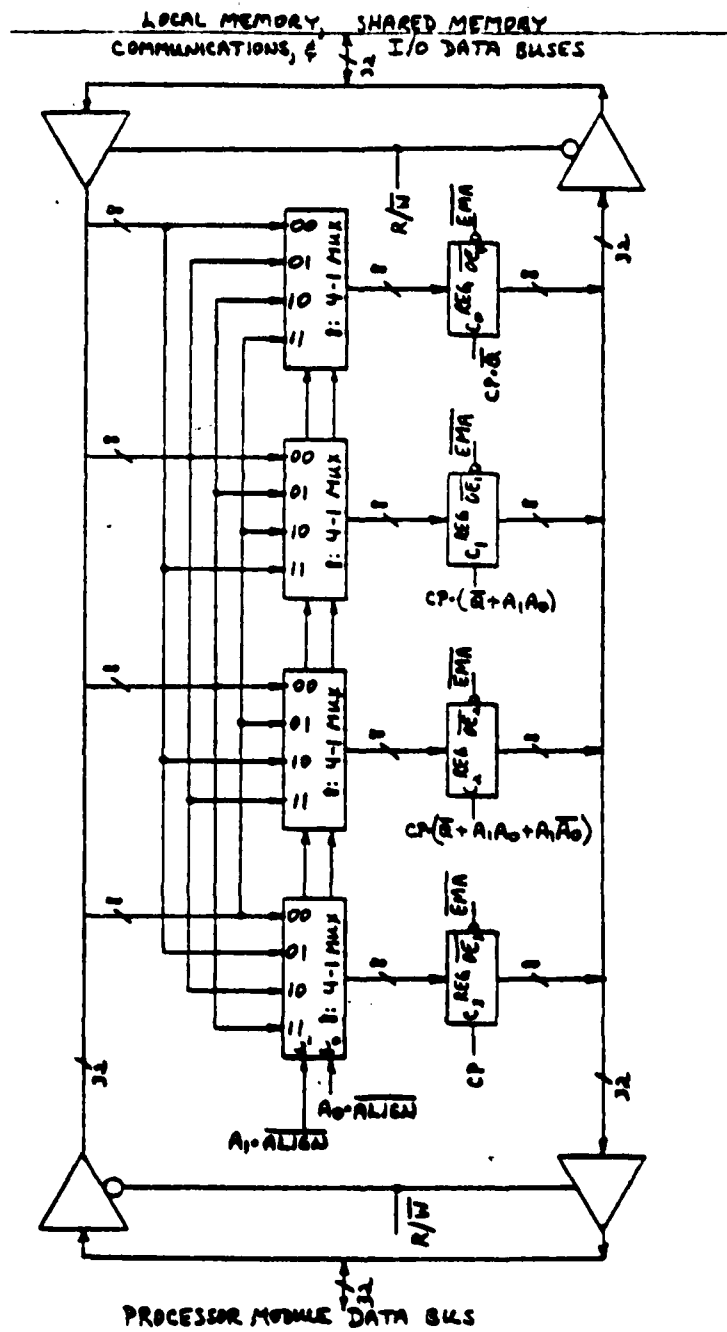


Figure 53. Data Rearrangement During Both Read and Write Operations Using the Memory-Alignment Unit

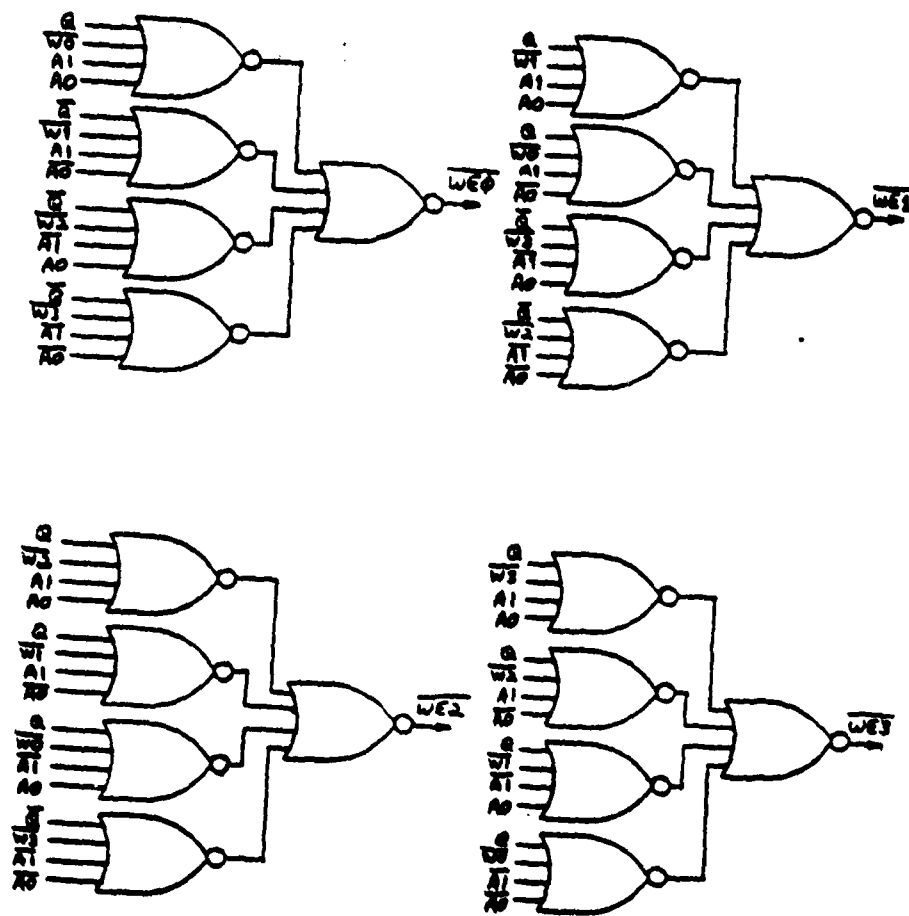


Figure 54. Hardware Logic for RAM Write Enables

tion set. As noted in Table 4, the best case average instruction execution time is 2.673×10^{-6} sec. (374,111 instruction/sec.) and the worst case average time is 8.807×10^{-6} sec. (113,546 instruction/sec.)

The details for the computation of individual instruction times are given in Appendix E. As noted there, the major reason for the variability on instruction times is the use of different address modes. The execution time for most instructions cannot be determined until the addressing mode used to obtain each operand is specified. The register direct modes require the least amount of time whereas indexed, indirect addressing modes require the most. For example, a longword source operand can be obtained in 0.2×10^{-6} sec. using register direct mode; on the other hand, the use of indexed longword byte-displacement deferred addressing requires 3.0×10^{-6} sec. if memory references are aligned and 3.4×10^{-6} sec. if not aligned. In Table 4, the best case execution time of an operand instruction is based on register direct addressing for each operand required for that instruction: the worst case is based on the use of the addressing mode which requires the largest amount of time to obtain each needed operand.

The microcode to emulate all the VAX 11/780 addressing modes is listed in Appendix D, from which the execution time of each mode may be obtained if desired. The definitions and functions of these addressing modes is available in Digital Equipment Corporation literature¹.

There are other factors which influence the execution time of certain instructions. In the case of instructions involving conditional branches, the execution time depends on whether or not an offset is added to the program counter. In addition, the execution times of all floating point instructions are data dependent. For example, in a floating-point addition the exponents of the two operands must be equalized by shifting the mantissa of one of the operands. The number of shifts depends on the data supplied. Normalization of the results of floating-point operations is another data dependent operation. The best case and worst case execution times for floating-point instructions are based on the variabilities of both data and addressing mode dependencies.

1. VAX 11/780 Architecture Handbook, Volume 1, Digital Equipment Corporation, Maynard, Massachusetts, 1977, pp. 5-3 through 5-33.

TABLE 4

INSTRUCTION TIMES AND USAGE FACTORS

| <u>Instruction Type</u> | <u>Usage Factor</u> | <u>Best Case (10⁻⁶sec.)</u> | <u>Worst Case (10⁻⁶ sec.)</u> |
|-------------------------------------|-------------------------|--|--|
| LOAD | .260 | 1.6 | 1.8 |
| LOAD-Double Precision | .001 | 2.2 | 2.8 |
| STORE | .188 | 1.6 | 1.8 |
| STORE-Double Precision | .001 | 2.2 | 2.6 |
| ADD/SUBTRACT | .028 | 1.6 | 11.2 |
| ADD/SUBTRACT-Floating Point | .067 | 5.8 | 37.6 |
| MULTIPLY | .001 | 8.0 | 17.6 |
| MULTIPLY-Floating Point | .068 | 11.0 | 37.8 |
| DIVIDE | .001 | 9.6 | 19.2 |
| DIVIDE-Floating Point | .007 | 12.4 | 76.0 |
| LOGICAL | .060 | 1.6 | 11.2 |
| SHIFT-5 Places | .009 | 4.4 | 10.8 |
| COMPARE | .051 | 1.4 | 7.8 |
| BRANCH | .218 | 0.4 | 1.0 |
| INDEX | .004 | 3.4 | 16.6 |
| REG-TO-REG Transfer | .000 | 1.0 | 1.0 |
| MISC | .029 | 1.2 | 7.6 |
| INPUT-OUTPUT (Set-up + 10 words) | .007 | 46.2 | 50.2 |

AVERAGE INSTRUCTION TIME

BEST CASE: 2.673×10^{-6} sec./instructionWORST CASE: 8.807×10^{-6} sec./instruction

SECTION V

CONCLUSIONS

The detailed investigation and analysis to derive a control algorithm for a trainer computer system consisting of a series of N microcomputers has been completed. The investigation included the following, as required by the specification:

- a) A study of the preferred algorithm
- b) Investigation of optimal combinations of hardware, firmware, and software for implementation of the control algorithm concept.
- c) Microcomputer characteristics analysis
- d) Microcomputer instruction synthesis
- e) Microcomputer performance analysis
- f) Common memory requirements analysis
- g) Problem partitioning.
- h) Algorithm implementation trade-offs

THE PREFERRED ALGORITHM

A literature search was conducted to determine alternative methods to the preferred algorithm. The resulting techniques were compared and the conclusion was drawn that the preferred algorithm was a viable approach that appeared to be better suited to present technology than other techniques examined.

The preferred algorithm was then examined to determine what criteria were necessary for successful implementation. The following prerequisites were established.

- a) The problem must be partitioned into disjoint tasks.
- b) Some mechanism must exist at run time to insure that parameters are passed in such a way as to insure that system precedence is maintained.

CONTROL ALGORITHM INVESTIGATION

A paper implementation of the control algorithm, based on the preferred algorithm and using a combination of hardware, firmware, and software, was designed. During the initial portion of the design several design guidelines were agreed upon:

- a) Distributed control should be used to the maximum extent possible to reduce system overhead.
- b) All major interfaces between major hardware blocks should be through a virtual machine structure to reduce the amount of hardware dependence.

- c) The major limitation on system performance is the shared memory bus since the processing power can always be increased by adding more processor modules. Therefore the design should minimize bus traffic where possible.

The major innovations of the design were:

- a) Use of the Application Task Manager (ATM) in each module to distribute much of the control function to the processor modules and to implement the virtual machine interface.
- b) Use of separate communication and control buses to enhance the concurrency of the system and the bandwidth of the shared memory bus.
- c) Use of the distributed cache shared memory to further enhance the bandwidth of the shared memory.

THE MICROCOMPUTER MODULES

The microcomputer module was designed for maximum performance while retaining maximum reliability. The 2900-series of bit-slice components were chosen for this design because their large-scale integration resulted in a minimum of components and their extensive instruction set yielded high performance. By choosing a microprogrammable design the combinational logic for control was reduced while still allowing design flexibility. To this end, the microprogram itself was written in independent modules which were then easily tested, modified, and corrected as necessary. Subroutines were used for the addressing modes thus allowing each instruction to use any addressing mode. This resulted in a very powerful instruction set.

The performance is a function of the microcycle instruction time and the number of microcycles per machine instruction. The microcycle time has been minimized by the use of a nonencoded horizontal microcode word and high-speed TTL bit-slice components. The inherent power of the 2900-series components minimizes the number of microcycles required. Also, the microcode has been optimized by instruction overlapping wherever possible. The resultant microcomputer has an average instruction execution time of 2.67 microseconds (best case) for a representative mix of instructions.

PARTITIONING THE APPLICATION PROBLEM

The key to the capabilities of the multiple microcomputer system resides in the partitioning of a simulation problem into a set of independent modules that can be executed in parallel. Four partitioning approaches to achieve parallelism have been examined, as follows:

- a) acyclic task graphs
- b) parallelism at the level of individual operations in assignment statements

- c) nearly-decomposable systems
- d) mathematical optimization

Of these four partitioning approaches, mathematical optimization and the concept of nearly-decomposable systems appear to have the most promise for the trainer simulation problem. After the problem is partitioned using one of these approaches, the speed-up factor developed in Section II yields a measure of the efficiency of the multiple microcomputer system.

SUMMARY

The multiple microcomputer system has been designed with a functionally distributed, hierarchical structure that allows sophisticated real-time trainers to be simulated efficiently and easily. The system utilizes distributed processing, distributed control, distributed input/output functions, and a distributed cache memory to implement a trainer that can be configured, modified, and maintained with minimum impact on users or programmers.

REFERENCES

1. Hansen, Per Brinch, Operating System Principles, NJ, Prentice-Hall, 1973, pp 60-76.
2. Agerwala, T., and Lint, B., "Communication in Parallel Computer Systems," Proc. 1978 Conf. Information Science and Systems, Johns Hopkins University, March 29-31, 1978, pp. 89-95.
3. Summer, C. F., Specification for Multiple Microcomputer Control Algorithm Investigation, NTEC N-74-105, May 22, 1978.
4. Bernstein, A. J., "Analysis of Programs for Parallel Processing," IEEE Trans. on Electronic Computers, Vol. EC-15, No. 5, Oct 1966, pp. 757-763.
5. Ramamoorthy, C. V. and Gonzalez, M. J., "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs," 1969 Fall Joint Computer Conference, AFIPS Conf. Proc., Vol. 35, Montvale, NJ: AFIPS Press, 1969, pp. 1-15.
6. Ramamoorthy, C. V., Chandy, K. M., and Gonzalez, M. J., "Optical Scheduling Strategies in a Multiprocessor System," IEEE Trans. on Computers, Vol. C-21, No. 2, Feb 1972, pp. 137-146.
7. Baker, K., "Improve Complex Software by Using Multiple Microprocessors," Microprocessors, Vol. 1, No. 3, Feb 1977, pp. 165-168.
8. Simon, H. A., The Science of the Artificial, Cambridge, MA: MIT Press, 1969.
9. Kuck, D. J., "Parallel Processing of Ordinary Programs," Advances in Computers (M. Rubinoff and M. C. Yovits, Ed's.), Vol. 15, NY: Academic Press, 1976, pp. 119-179.
10. Cylmer, S. J. and Price, P. E., "Partitioning Software for Advanced Simulation Computer Systems," to be published June 1979.
11. Liu, C. L., Introduction to Combinatorial Mathematics, NY: McGraw-Hill, 1968, pp. 38-40.
12. Wilf, H. S. and Nijenhuis, A., Combinatorial Algorithms, NY: Academic Press, 1975, pp. 110-117.
13. Kober, R., "A Fast Communication Processor for the SMS Multimicroprocessor System," Second Symposium on Micro-Architecture (Sami, M., Wilmink, J., and Zaks, R., Ed's.), North-Holland Publ. Co., 1976, pp. 183-189.
14. Fung, K. T., and Torng, H. C., "On the Analysis of Memory Conflicts and Bus Contentions in a Multi-Microprocessor System," IEEE Trans. on Computers, Vol. C-27, No. 1, Jan 1979, pp. 28-37.

NAVTRAEQUIPCEN 78-C-0157-1

15. Reyling, George, Jr., "Performance and Control Of Multiple Microprocessor Systems," Computer Design, March 1974, pp. 81-86.
16. VAX 11/780 Architecture Handbook, Volume 1, Digital Equipment Corporation, Maynard, MA, 1977, pp. 5-3 through 5-33.

APPENDIX A

ATM COMMUNICATION STATE

Shown in Figure A-1 is the memory assignment of the control bus address space. Each of the N processors is assigned a block of M spaces on the control bus. These M spaces are used for a data port and a status and control register. The actual number of address spaces, M, is implementation dependent. The control bus is a synchronous bus having the following signals:

- a) Address lines (A_0-A_i)
- b) Data lines (D_0-D_j)
- c) Read/write line (R/\bar{W})
- d) Enable line (\overline{CBEN})
- e) Synchronization line (SYNC)
- f) Slow-memory line (READY), optional
- g) Interrupt lines (CBIRQ, CBNMI)

The control processor communicates with a given application processor by use of the appropriate address. The actual communication of messages on the bus is asynchronous. The handshake signals required for communication on the bus are accessed through the status and control register of the chosen application processor. The signals are CPSYN (control processor sync) and APSYN (application processor sync). The control processor communicates with an application processor by writing a word into the data port, then asserting the CPSYN signal. The application processor acknowledges receiving the data by asserting the APSYN signal at which time the control processor clears CPSYN, followed by the application processor clearing APSYN and the transaction is complete. The same technique may be used when the application processor sends information to the control processor (except that the roles of APSYN and CPSYN are reversed), or the control processor may interrogate the application processor. The status and control register also contains an end-of-transmission bit (EOT) which is bidirectional and is used to indicate that the message is complete.

Only the control processor may be the master of the control bus. Individual application processors may send data to their data port, not to the actual bus. This prevents an application processor from blocking the bus. In addition

it allows the control processor to have more than one transaction in progress at one time.

The first byte of any message passed on the control bus is always an opcode. The opcode may be followed by as many operands as are required. In some instances a string of operands may be followed by a terminator byte. One value of the opcode (all 1's) is reserved as an escape for future expansion.

The COMMUNICATION state opcodes are divided into three major groups: processor-level activities, task-level activities, and test/debug activities. A listing of COMMUNICATION state operations is given below.

1.0 PROCESSOR-LEVEL OPERATIONS

- 1) Enter HALT state
- 2) Enter WAIT state
- 3) Write into processor memory
- 4) Read processor memory

2.0 TASK-LEVEL OPERATIONS

- 1) Load task control blocks
- 2) Load system program
- 3) Load system program queue
- 4) Load application programs
- 5) Load jobs queue
- 6) Read task control blocks
- 7) Read system program queue
- 8) Read jobs queue
- 9) Read ready task queue
- 10) Read currently active program

3.0 TEST/DEBUG OPERATIONS

- 1) Set breakpoints
- 2) Read breakpoints
- 3) Clear breakpoints
- 4) Enter single-step mode
- 5) Exit single-step mode
- 6) Pulse single-step mode

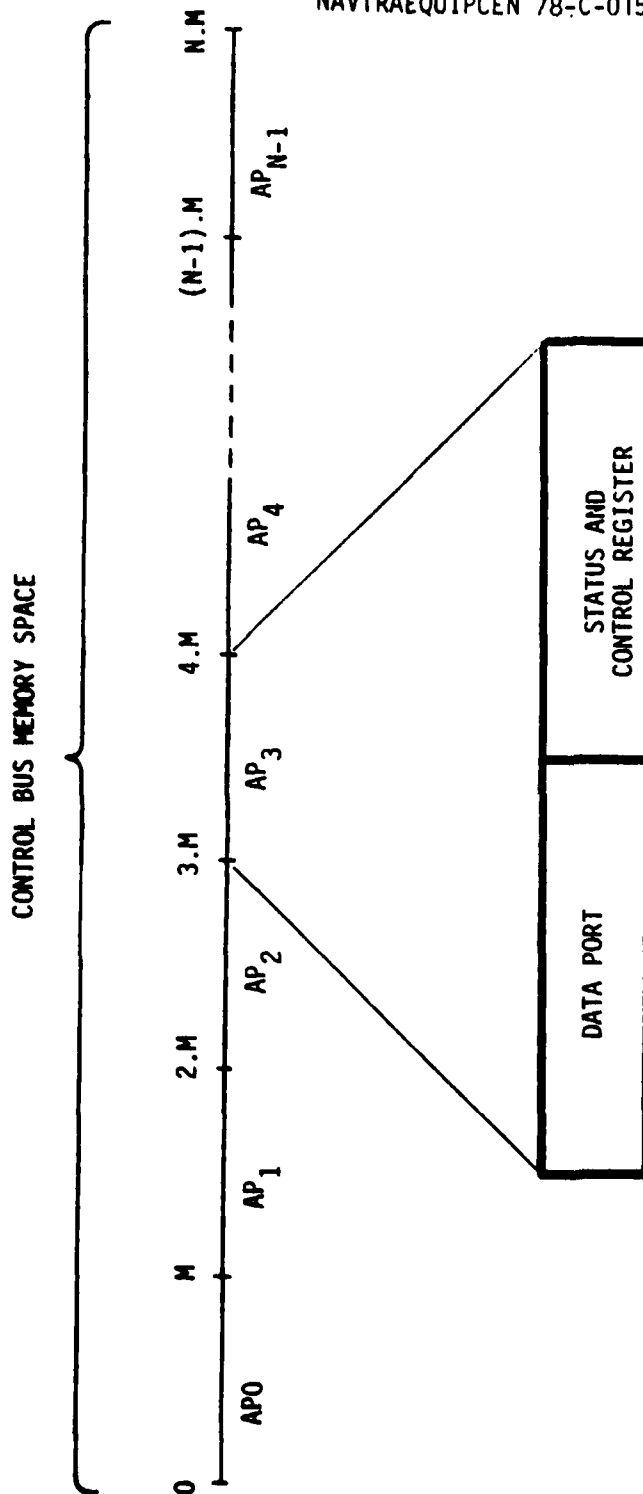


Figure A-1. Assignment of control bus memory space to the application processors

APPENDIX B

SUPERVISOR CALLS

1. TASK MANAGEMENT
 - a) Get task priority
 - b) Get task time limit
 - c) Set task priority*
 - d) Set task time limit*
2. FLAG MANAGEMENT
 - a) Get flags
 - b) Set flags
 - c) Assign flags*
 - d) Deassign flags*
3. INTERRUPT CONTROL
 - a) Assign interrupts**
 - b) Mask interrupts**
4. TASK CONTROL
 - a) Initiate task* (if dormant)
 - b) Initiate task* (queue if active or suspended)
 - c) Initiate task* at specific time
 - d) Initiate task* after specific interval
 - e) Terminate task*
 - f) Terminate task* at specific time
 - g) Terminate task* after specific interval
 - h) Suspend task* (same as terminate)
 - i) Restart task* (same as terminate)

All above commands may be conditional. The conditions are:

- a) Flag state
- b) AND flags
- c) OR flags
- j) Remove task**
- k) Insert task**

5. TIME MANAGEMENT
 - a) Get time
(Set time can only be done by control processor)
6. ERROR HANDLING
 - a) Log error locally
 - b) Report error to control processor
 - c) Log error locally and terminate task
 - d) Report error to control processor and suspend task
7. RESOURCE ALLOCATION
 - a) Request resource
 - b) Release resource
8. I/O AND MESSAGE SERVICES
 - a) Send message (to task)
 - b) Read message (from task)
 - c) Send message (to device)
 - d) Read message (from device)
 - e) Send string** (to shared memory)
 - f) Read string** (from share memory)
9. EVENT CONTROL
 - a) Signal operation
 - b) Wait operation
 - c) Proceed on event
10. MEMORY MANAGEMENT
 - a) Assign task memory boundaries**

SVC's marked with a single asterisk (*) require that the asserting task be of higher priority than the target task where they are different. SVC's marked with a double asterisk (**) may only be used by system programs.

NAVTRAEQUIPCEN 78-C-0157-1

APPENDIX C

VAX-Emulator Definition File

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 1 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

0010 TITLE VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

0020 SIZE 110
0030 OPT D=UFTA,DT:1,N=120,M=L=PROJECT.AL:1
0040

0050 : *****
0060 : * DEFINITION FILE OF ALU CONTROL UNIT *
0070 : *****
0080

0090 :
0100 : ALU CONTROL UNIT DEFINITION FILE INCLUDES THE DEFINITION
0110 : OF ALL CONCERNED VARIABLES, AND OPERATIONS.
0120 :
0130 : * ALU 64X,ASRC,AADR,BSRC,BADR,FUNT,DEST,WREN,CYIN,SHIFFT,QISHFT
0140 :
0150 ALU FORM 64X,3VQ0,6VH10,1VB0,6VH0,4VH0,6VQ60,5VH0,3VQ1,9X,2VQ0,2VQ0,1X
0160

0170 :
0180 : * ALU A-SOURCE VARIABLES (ASRC):
0190 :
0200 REG EQU 00 : 2905 DPR AND 2903 RAM
0210 DIR EQU 10 : DATA INPUT REGISTER
0220 ADR EQU 20 : ADDRESS BUFFER
0230 BSB EQU 30 : BYTE SHIFT BUFFER
0240 CONST EQU 40 : CONSTANTFROM PIPELINE REGISTER
0250 DTYPE EQU 50 : DATA TYPE REGISTER
0260 LITERAL EQU 60 : DATAFROM LITERAL INPUT
0270 NOSRC EQU 70 : NO EXTERNAL SOURCE
0280

0290 :
0300 : * ALU A-ADDRESS AND B-ADDRESS (AADR & BADR):
0310 :
0320 R0 EQU 6H00 : 2903 INTERNAL RAM, USER REG.
0330 R1 EQU 6H01 : 2903 INTERNAL RAM, USER REG.
0340 R2 EQU 6H02 : 2903 INTERNAL RAM, USER REG.
0350 R3 EQU 6H03 : 2903 INTERNAL RAM, USER REG.
0360 R4 EQU 6H04 : 2903 INTERNAL RAM, USER REG.
0370 R5 EQU 6H05 : 2903 INTERNAL RAM, USER REG.
0380 R6 EQU 6H06 : 2903 INTERNAL RAM, USER REG.
0390 R7 EQU 6H07 : 2903 INTERNAL RAM, USER REG.
0400 R8 EQU 6H08 : 2903 INTERNAL RAM, USER REG.
0410 R9 EQU 6H09 : 2903 INTERNAL RAM, USER REG.
0420 R10 EQU 6H0A : 2903 INTERNAL RAM, USER REG.
0430 R11 EQU 6H0B : 2903 INTERNAL RAM, USER REG.
0440 R12 EQU 6H0C : 2903 INTERNAL RAM, USER REG.
0450 R13 EQU 6H0D : 2903 INTERNAL RAM, USER REG.
0460 R14 EQU 6H0E : 2903 INTERNAL RAM, STACK POINTER
0470 SP EQU R14
0480 R15 EQU 6H0F : 2903 INTERNAL RAM, PROGRAM COUNTER
0490 PC EQU R15
0500 :

NAVTRAEQUIPCEN 78-C-0157-1

```

PAGE      2  PROJECT  MASH-DEFINITION  VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

0510 SRC      EQU      6H10          ; DPR WORKING REG., FOR SOURCE OPERAND IN JSMD.
0520 SRC2     EQU      6H11          ; DPR WORKING REG.
0530 SRC3     EQU      6H12          ; DPR WORKING REG.
0540 SRC4     EQU      6H13          ; DPR WORKING REG.
0550 DST      EQU      6H14          ; DPR WORKING REG., FOR DESTINATION OPERAND IN JMD.
0560 DST2     EQU      6H15          ; DPR WORKING REG.
0570 EA       EQU      6H16          ; DPR WORKING REG., EFFECTIVE ADDRESS
0580 LENGTH   EQU      6H17          ; DPR WORKING REG.
0590 INDEX    EQU      6H18          ; DPR WORKING REG., FOR INDEX ADDRESSING
0600 INDEX2   EQU      6H19          ; DPR WORKING REG.
0610 INDEX3   EQU      6H1A          ; DPR WORKING REG.
0620 INDEX4   EQU      6H1B          ; DPR WORKING REG.
0630 WRI      EQU      6H1C          ; DPR WORKING REG.
0640 EIGHT    EQU      6H1D          ; CONSTANT REG. FOR VALUE 8
0650 FOUR     EQU      6H1E          ; CONSTANT REG. FOR VALUE 4
0660 TWO      EQU      6H1F          ; CONSTANT REG. FOR VALUE 2
0670 ;
0680 AMODE    EQU      6H20          ; REG. ADDRESS SPECIFIED BY VAX COUNTER
0690 ;

0700 ;
0710 ; 6 ALU B-SOURCE SELECT VARIABLES (BSRC) :
0720 ;
0730 Q        EQU      1B           ; FROM Q REGISTER
0740 ;

0750 ;
0760 ; 6 DEFINITION OF THE FUNCTION FIELD (FUNC):
0770 ;
0780 ;
0790 SUBSR     EQU      01H          ; S-R-1-CIN
0800 SUB       EQU      02H          ; R-S-1-CIN
0810 ADD       EQU      03H          ; R+S+CIN
0820 ADDS      EQU      04H          ; S+CIN
0830 ADDSN     EQU      05H          ; (-S)+CIN
0840 ADDR      EQU      06H          ; R+CIN
0850 ADDRN     EQU      07H          ; (-R)+CIN
0860 ZERO      EQU      08H          ; 0, NO OPERATION
0870 ANDRN     EQU      09H          ; (R/) AND S
0880 EXNOR     EQU      0AH          ; R EXCLUSIVE NOR S
0890 EXOR      EQU      0BH          ; R EXCLUSIVE OR S
0900 AND       EQU      0CH          ; R AND S
0910 NOR       EQU      0DH          ; R NOR S
0920 NAND      EQU      0EH          ; R NAND S
0930 OR        EQU      0FH          ; R OR S
0940 ;

0950 ;
0960 ; 6 DEFINITION OF THE DESTINATION2 FIELD (DEST):
0970 ;
0980 ;
0990 FAD       EQU      000          ; Y=ARITH F/2, SHIFTRIGHT
1000 FLB      EQU      070          ; Y=LOG F/2.
1010 FADQ     EQU      100          ; Y=ARITH F/2, Q=LOG Q/2
1020 FLBQ     EQU      170          ; Y=LOG F/2, Q=LOG Q/2

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 3 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.3, 11/02/79

```

1030 PAR      EQU 200      ; Y=F, S100=PAR
1040 PQD      EQU 270      ; Y=F, S100=PAR, Q=LOG(Q/2), (W/)=H
1050 PFD      EQU 300      ; Y=F, S100=PAR, Q=F, (W/)=H
1060 PLQ      EQU 370      ; Y=F, S100=PAR, Q=F, (W/)=L
1070 FAU      EQU 400      ; Y=ARITH 2F, LEFTSHIFT 1 BIT
1080 FLU      EQU 470      ; Y=LOG 2F
1090 FAUQ     EQU 500      ; Y=ARITH 2F, Q=LOG 2Q
1100 FLUQ     EQU 570      ; Y=LOG 2F, Q=LOG 2Q
1110 F        EQU 600      ; Y=F, (W/)=H
1120 QU       EQU 670      ; Y=F, Q=LOG 2Q, (W/)=H
1130 ;
1140 EXTB     EQU 710      ; EXTEND BYTE TO LONGWORD
1150 EXTW     EQU 730      ; EXTEND WORD TO LONGWORD
1160 EXTBW    EQU 750      ; EXTEND BYTE TO WORD
1170 ;
1180 ;

1190 ;
1200 ; % DEFINITION OF DUAL PORT REGISTERS WRITE ENABLE (WREN):
1210 ;
1220 WNR      EQU 5H00      ; NO WRITE
1230 R0       EQU 5H01      ; ONLY BYTE 0 WRITED
1240 R1       EQU 5H02      ; ONLY BYTE 1 WRITED
1250 WRD      EQU 5H03      ; WORD WRITTEN.
1260 R2       EQU 5H04      ; ONLY BYTE 2 WRITED
1270 R3       EQU 5H08      ; ONLY BYTE 3 WRITED
1280 UWRD     EQU 5H0C      ; UPPER WORD WRITTEN.
1290 R321     EQU 5H0E      ; THE UPPER THREE BYTES WRITED
1300 LWRD     EQU 5H0F      ; LONG WORD WRITTEN
1310 INS      EQU 5H10      ; WRITE ENABLE SPECIFIED BY INSTRUCTION
1320 ;
1330 ;

1340 ;
1350 ; % DEFINITION OF CARRY INPUT TO ALUO (CYIN):
1360 ;
1370 CZERO     EQU 00       ; 0 INPUTED AS CARRY
1380 CONE      EQU 10       ; 1 INPUTED AS CARRY
1390 CY        EQU 20       ; FROM CARRY FLAG
1400 CYN       EQU 30       ; FROM INVERTED CARRY FLAG (BORROW)
1410 ZIN       EQU 40       ; FROM ZERO FLAG
1420 ;
1430 ;

1440 ;
1450 ; % DEFINITION OF SHIFT MUX-SELECT (SIGN & QSIGN):
1460 ;
1470 SMT0      EQU 00       ; SELECT 0 AS INPUT
1480 SMT1      EQU 10       ; SELECT 1 AS INPUT, EXCEPT S103
1490 S10N      EQU 10       ; SELECT SIGN FF TO S1031
1500 S10       EQU 20       ; SELECT S10 AS INPUT
1510 Q10       EQU 30       ; SELECT Q10 AS INPUT
1520 S8FF0     EQU 20       ; SELECT SIGN FF 00 AS INPUT
1530 S8FF1     EQU 30       ; SELECT SIGN FF 02 AS INPUT
1540 ;

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 4 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

1550

1560 ;

1570 ; * FLOATING POINT SIGN FLIP-FLOP CONTROL

1580 ; IN FLOATING POINT ARITHMETIC THERE NEEDS SPECIAL SIGN FLIP-FLOPS.

1590 ;

1600 FSIGN FORM 20X.2UB01.90X

1610 ;

1620

1630 ;

1640 ; * DEFINITION OF FLOATING POINT SIGN FLIP-FLOP VARIABLES:

1650 ;

1660 SIGN0 EQU 2B01 ; SIGN FF #0

1670 SIGN1 EQU 2B10 ; SIGN FF #1

1680 ;

1690 EJECT

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 5 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 3.5, 11-02/79

```
1700 : *****
1710 : * DEFINITION FILE OF CONTROL SEQUENCER *
1720 : *****
1730 :
```

```
1740 :
1750 : THIS FILE DEFINES THE CONTROL SEQUENCES OF MICROPROGRAMMING
1760 :
1770 :
1780 :
1790 : * UNCONDITIONAL JUMP
1800 : JUMP, LOCATION
1810 :
1820 JUMP      FORM 33X, 3B100, 4H3, 16VHT, 1B0, 1X, 1B1, 5B01111, 4BX
1830 :
```

```
1840 :
1850 : * CONDITIONAL JUMP:
1860 : CJMP, LOCATION, CC
1870 :
1880 CJMP      FORM 33X, 3B100, 4H3, 16VHT, 1B0, 1X, 6VB, 4BX
1890 :
```

```
1900 :
1910 : * JUMP TO SUBROUTINE :
1920 : JSB, LOCATION
1930 :
1940 JSB      FORM 33X, 3B100, 4H1, 16VHT, 1B0, 1X, 1B1, 5B01111, 4BX
1950 :
```

```
1960 :
1970 : * CONDITIONAL JUMP TO SUBROUTINE :
1980 : CJSB, LOCATION, CC
1990 :
2000 CJSB     FORM 33X, 3B100, 4H1, 16VHT, 1B0, 1X, 6VB, 4BX
2010 :
```

```
2020 :
2030 : * JUMP TO MAP ( OR INTERRUPT HANDLER ROUTINES ) :
2040 : JMAP
2050 JMAP      FORM 33X, 3B001, 4H7, 16X, 1B0, 1X, 1B0, 5B01100, 4BX
2060 :
```

```
2070 :
2080 : CONDITIONAL JUMP TO MAP :
2090 : CJMP, CC
2100 :
2110 CJMAP     FORM 33X, 3B001, 4H3, 16X, 1B0, 1X, 6VB, 4BX
2120 :
```

```
2130 :
2140 : * JUMP TO SOURCE MODE ROUTINES :
2150 : JSMD
2160 :
2170 JSMD      FORM 33X, 3B010, 4H1, 16X, 1B0, 1B1, 1B1, 5B01111, 4BX
2180 :
```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 5 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

```

2190 :
2200 : * CONDITIONAL JUMP TO SOURCE MODE ROUTINES :
2210 : CJSMD.CC
2220 :
2230 CJSMD      FORM 33X,3B010.4M1.16X,1B0,1B1.6VB.48X
2240 :

2250 :
2260 : * JUMP TO DESTINATION MODE ROUTINES :
2270 : JDMD
2280 :
2290 JDMD      FORM 33X,3B010.4M3.16X,1B0,1B0,1B1,5B01111.48X
2300 :

2310 :
2320 : * CONDITIONAL JUMP TO DESTINATION MODE ROUTINES :
2330 :CJDMD.CC
2340 :
2350 CJDMD     FORM 33X,3B010.4M3.16X,1B0,1B0.6VB.48X
2360 :

2370 :
2380 : * RETURN FROM SUBROUTINES :
2390 : RET
2400 :
2410 RET      FORM 33X,3B000.4MA.16X,1B0,1X,1B1,5B01111.48X
2420 :

2430 :
2440 : * CONDITIONAL RETURN FROM SUBROUTINES :
2450 :CRET.CC
2460 :
2470 CRET     FORM 33X,3B000.4MA.16X,1B0,1X.6VB.48X
2480 :

2490 :
2500 : * CONTINUE TO PROCEED THE NEXT MICROINSTRUCTION :
2510 : CONT
2520 CONT     FORM 36X.4HE.72X
2530 :

2540 :
2550 : * PIPELINE CONSTANT GENERATION :
2560 : PIPE.CONSTANT(HEX)
2570 :
2580 PIPE     FORM 40X,16VMT.56X
2590 :

2600 :
2610 : * REPEAT PIPELINE REGISTER :
2620 :
2630 REPPL    FORM 33X,3B100.4M9.16VMT,1B1.55X
2640 :

2650 :
2660 : * CONDITIONAL JUMP PIPELINE & POP :
2670 : CJPP.LOCATION.CC
2680 :

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 7 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

2690 CJPP FORM 33X,3B100.4MB,16VMT,1B0,1X,6VB,4BX
2700

2710 ;
2720 ; * PUSH & CONDITIONAL LOAD COUNTER :
2730 ; PUSH,NUMBER,CC

2740 ;
2750 PUSH FORM 33X,3B100.4MB,16VMT,1B0,1X,6VB,4BX
2760

2770 ;
2780 ; * REPEAT LOOP :
2790 ;
2800 REPOP FORM 33X,3B000.4MB,16X,1B1,55X
2810

2820 ;
2830 ; * LOAD COUNTER & CONTINUE :
2840 ; LDCT,NUMBER
2850 ;
2860 LDCT FORM 33X,3B100.4MB,16VMT,1B1,55X
2870

2880 ;
2890 ; * REPEAT PIPELINE, IF COUNTER NEQ 0 :
2900 ;
2910 RPCT FORM 33X,3B100.4MB,16X,1B1,55X
2920

2930 ;
2940 ; * TEST THE END OF THE LOOP :
2950 ; LOOP,CC
2960 ;
2970 LOOP FORM 33X,3B000.4MB,16X,1B1,1X,6VB,4BX
2980

2990 EJECT

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 8 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 3.5, 11/02/79

```

3000 ;
3010 ; *****
3020 ; * DEFINITIONS OF THE CONDITION CODE VARIABLES *
3030 ; *****
3040 ;

```

```

3050 ;
3060 ; CONDITION CODE DEFINES THE TEST CONDITIONS FOR SEQUENCING.
3070 ;
3080 ;

```

```

3090 EQU      EQU      6B100000      ; Z EQU 1: ZERO
3100 NEQ      EQU      6B000000      ; Z EQU 0: NOT ZERO
3110 ;
3120 LSS      EQU      6B100001      ; N EQU 1: NEGATIVE
3130 GEQ      EQU      6B000001      ; N EQU 0: NOT NEGATIVE.
3140 ;
3150 VS       EQU      6B100010      ; V EQU 1: OVERFLOW.
3160 VC       EQU      6B000010      ; V EQU 0: NOT OVERFLOW.
3170 ;
3180 CS       EQU      6B100011      ; C EQU 1: CARRY.
3190 LSSU     EQU      CS            ; LESS THAN UNSIGNED.
3200 ;
3210 CC       EQU      6B000011      ; C EQU 0: NO CARRY.
3220 GEQU     EQU      CC            ; GREATER THAN OR EQU UNSIGNED.
3230 ;
3240 LEQ      EQU      6B100100      ; N OR Z EQU 1: <= 0
3250 GTR      EQU      6B000100      ; N AND Z EQU 0: > 0
3260 ;
3270 LEQU     EQU      6B100101      ; C OR Z EQU 1
3280 GTRU     EQU      6B000101      ; C AND Z EQU 0.
3290 ;
3300 BYT1     EQU      6B100110      ; INSTRUCTION TYPE OF BYTE.
3310 NB1      EQU      6B000110      ; NOT ONE BYTE.
3320 ;
3330 BYT2     EQU      6B100111      ; INSTRUCTION TYPE OF WORD.
3340 NB2      EQU      6B000111      ; NOT WORD.
3350 ;
3360 BYT4     EQU      6B101000      ; INSTRUCTION TYPE OF LONGWORD.
3370 NB4      EQU      6B001000      ; NOT LONGWORD.
3380 ;
3390 BYT8     EQU      6B101001      ; INSTRUCTION TYPE OF EIGHT BYTES.
3400 NB8      EQU      6B001001      ; NOT EIGHT BYTES.
3410 ;
3420 FLOAT    EQU      6B101010      ; DATA TYPE OF FLOATING.
3430 NFLOAT   EQU      6B001010      ; NOT FLOATING.
3440 ;
3450 ALIGN    EQU      6B101011      ; ADDRESS ALIGNED.
3460 NALIGN   EQU      6B001011      ; ADDRESS NOT ALIGNED.
3470 ;
3480 INT      EQU      6B101100      ; INTERRUPT PENDING.
3490 NOINT    EQU      6B001100      ; NO INTERRUPT.
3500 ;
3510 EAO      EQU      6B101101      ; EFFECTIVE ADDRESS ONLY.
3520 NEAO     EQU      6B001101      ; NOT EFFECTIVE ADDRESS.
3530 ;
3540 BITO     EQU      6B101110      ; ALU BITO EQU 1.

```

NAVTRAEQUIPCEN 78-C-0157-1

| PAGE | 9 | PROJECT | MASH-DEFINITION | VAX-EMULATOR DEFINITION | PHASE, REV 5.5, 11/02/79 |
|------|----------|---------|-----------------|-------------------------|------------------------------------|
| 3550 | NBIT0 | EQU | 6B001110 | | : ALU BIT0 EQU 0. |
| 3560 | : | | | | |
| 3570 | TRUE | EQU | 6B101111 | | : LOGIC 1. |
| 3580 | FALSE | EQU | 6B001111 | | : LOGIC 0. |
| 3590 | : | | | | |
| 3600 | MORE0 | EQU | 6B110000 | | : MANTISSA OR EXPONENT EQU 0. |
| 3610 | NMORE0 | EQU | 6B010000 | | : MANTISSA AND EXPONENT NOT EQU 0. |
| 3620 | : | | | | |
| 3630 | ALU0 | EQU | 6B110001 | | : ALU OUTPUT EQU 0. |
| 3640 | NALU0 | EQU | 6B010001 | | : ALU OUTPUT NOT EQU 0. |
| 3650 | : | | | | |
| 3660 | SIO15 | EQU | 6B110010 | | : SIO15 OUTPUT EQU 1. |
| 3670 | NSIO15 | EQU | 6B010010 | | : SIO15 OUTPUT EQU 0. |
| 3680 | : | | | | |
| 3690 | INDXMOD | EQU | 6B110011 | | : INDEX ADDRESSING MODE. |
| 3700 | NINDXMOD | EQU | 6B010011 | | : NOT INDEX ADDRESSING MODE. |
| 3710 | : | | | | |
| 3720 | BIT31 | EQU | 6B110100 | | : ALU BIT31 EQU 1. |
| 3730 | NBIT31 | EQU | 6B010100 | | : ALU BIT31 EQU 0. |
| 3740 | : | | | | |
| 3750 | CONTIN | EQU | 6B110101 | | : CC FOR CONTINUING. |
| 3760 | NCONTIN | EQU | 6B010101 | | : NOT CONTINUING. |
| 3770 | EJECT | | | | |

NAVTRAEQUIPCEN 78-C-0157-1

| PAGE | 9 | PROJECT | NASM-DEFINITION | VAX-EMULATOR DEFINITION PHASE, REV 3.3, 11/02/79 |
|------|-----------|---------|-----------------|--|
| 3550 | NSITO | EQU | 68001110 | : ALU BITO EQU 0. |
| 3560 | : | | | |
| 3570 | TRUE | EQU | 68101111 | : LOGIC 1. |
| 3580 | FALSE | EQU | 68001111 | : LOGIC 0. |
| 3590 | : | | | |
| 3600 | MOREO | EQU | 68110000 | : MANTISSA OR EXPONENT EQU 0. |
| 3610 | NMOREO | EQU | 68010000 | : MANTISSA AND EXPONENT NOT EQU 0. |
| 3620 | : | | | |
| 3630 | ALUO | EQU | 68110001 | : ALU OUTPUT EQU 0. |
| 3640 | NALUO | EQU | 68010001 | : ALU OUTPUT NEG 0. |
| 3650 | : | | | |
| 3660 | SIOIS | EQU | 68110010 | : SIOIS OUTPUT EQU 1. |
| 3670 | NSIOIS | EQU | 68010010 | : SIOIS OUTPUT EQU 0. |
| 3680 | : | | | |
| 3690 | INDEXMOD | EQU | 68110011 | : INDEX ADDRESSING MODE. |
| 3700 | NINDEXMOD | EQU | 68010011 | : NOT INDEX ADDRESSING MODE. |
| 3710 | : | | | |
| 3720 | BIT31 | EQU | 68110100 | : ALU BIT31 EQU 1. |
| 3730 | NBIT31 | EQU | 68010100 | : ALU BIT31 EQU 0. |
| 3740 | : | | | |
| 3750 | CONTIN | EQU | 68110101 | : CC FOR CONTINUING. |
| 3760 | NCONTIN | EQU | 68010101 | : NOT CONTINUING. |
| 3770 | EJECT | | | |

NAVTRAEQUIPCEN 78-C-0157-1

| PAGE | 11 | PROJECT | MASH-DEFINITION | VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79 |
|------|-------|---------|-----------------|--|
| 4270 | BYTE | EQU | 4B0001 | ! TRANSFER ONLY ONE BYTE (BYTE 00). |
| 4280 | WORD | EQU | 4B0011 | ! TRANSFER ONLY ONE WORD. |
| 4290 | LWORD | EQU | 4B1111 | ! TRANSFER LONG WORD. |
| 4300 | BYTE1 | EQU | 4B0010 | ! WRITE ONLY BYTE 01. |
| 4310 | BYTE2 | EQU | 4B0100 | ! WRITE ONLY BYTE 02. |
| 4320 | BYTE3 | EQU | 4B1000 | ! WRITE ONLY BYTE 03. |
| 4330 | | EJECT | | |

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 12 PROJECT MASM-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 3.5, 11/02/79

```

4340 ;
4350 ; *****
4360 ; * DEFINITION FILE OF FLAGS CONTROL *
4370 ; *****
4380 ;

4390 ;
4400 ; THIS FILE DEFINES THE OPERATIONS OF ALU FLAGS WHICH INCLUDES
4410 ; V,N,Z AND C ALSO THE SOURCES OF THE FLAGS' CHANGE.
4420 ;
4430 ;

4440 ;
4450 ; * FLAGS CONTROL
4460 ; FLAGS,FLAGS-CHANGED,FLAGS-SOURCES
4470 ;
4480 FLAGS      FORM 98X,4080000,20800,20800,100,54
4490 ;

4500 ;
4510 ; * DEFINITION OF FLAGS VARIABLES :
4520 ;
4530 NONE      EQU 480000          ; NO FLAGS CHANGED.
4540 Z         EQU 480001          ; SET Z=1, ZERO
4550 N         EQU 480010          ; SET N=1, NEGATIVE.
4560 NZ        EQU 480011          ; SET N, Z=1
4570 V         EQU 480100          ; SET V=1, OVERFLOW.
4580 VZ        EQU 480101          ; SET V, Z=1.
4590 VN        EQU 480110          ; SET V, N=1.
4600 VNZ       EQU 480111          ; SET V,N,Z=1.
4610 C         EQU 481000          ; SET C=1, CARRY.
4620 CZ        EQU 481001          ; SET C,Z=1.
4630 CN        EQU 481010          ; SET C,N=1.
4640 CNZ       EQU 481011          ; SET C,N,Z=1.
4650 CV        EQU 481100          ; SET C,V=1.
4660 CVZ       EQU 481101          ; SET C,V,Z=1.
4670 CUN       EQU 481110          ; SET C,V,N=1.
4680 CUNZ      EQU 481111          ; SET ALL FLAGS.
4690 ALL       EQU CUNZ           ; SET ALL FLAGS.
4700 ;

4710 ;
4720 ; * DEFINITION OF FLAGS SOURCES (V,N,Z) :
4730 ;
4740 AF        EQU 2800            ; ALU FLAGS.
4750 DIRL      EQU 2801            ; DIRECTLY LOAD FROM ALU.
4760 ZER       EQU 2810            ; CLEAR FLAGS.
4770 BOR       EQU 2811            ; LOAD CARRY FROM ALU CARRY.
4780 ;

4790 ;
4800 ; * DEFINITION OF FLAGS SOURCES (NZ) :
4810 ;
4820 NZAF      EQU 180             ; NZ LOADED FROM ALU FLAGS.
4830 NZDIRL    EQU 181            ; NZ DIRECTLY LOADED.
4831 ;

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 13 PROJECT MASH-DEFINITION VAL-EMULATOR DEFINITION PHASE, REV 3.5, 11/02/79
4840 EJECT

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 14 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

```
4850 :
4860 : *****
4870 : * DEFINITION FILE OF BUS CONTROL *
4880 : *****
4890 :
```

```
4900 :
4910 : BUS CONTROL FILE DEFINES THE DATA TRANSFER BETWEEN
4920 : BUSES AND REGISTERS.
4930 :
4940 :
```

| | | |
|-----------|-------------------|--------------------------------|
| 4950 BPSL | FORM 24X,2810,84X | I DATA-BUS <- PSL. |
| 4960 BDDR | FORM 24X,2801,84X | I DATA-BUS <- DATA OUTPUT REG. |
| 4970 BPC | FORM 26X,2810,84X | I ADDRESS BUS <- PC. |
| 4980 BADR | FORM 26X,2801,84X | I ADDRESS BUS <- ADDRESS REG. |
| 4990 | EJECT | |

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 15 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

```

4991 : *****
5000 : * DEFINITION FILE OF REGISTERS, COUNTERS CONTROL *
5010 : *****
5020 :

```

```

5030 :
5040 : * REGISTER LOAD FORMAT
5050 :
5060 LEC      FORM  5X.1VB1.106X      : LOAD EXPONENT COUNTER.
5070 LPC      FORM  6X.1VB1.105X      : LOAD PC.
5080 LADDRD   FORM  7X.1VB1.104X      : LOAD ADDRESS REGISTER.
5090 LDOR     FORM  8X.1VB1.103X      : LOAD DATA OUTPUT REG.
5100 LPSL     FORM  9X.1VB1.102X      : LOAD PSL.
5110 LDIR     FORM  10X.1VB1.101X     : LOAD DATA-INPUT-REG FROM DATABUS
5120 LOPCR    FORM  11X.1VB1.100X     : LOAD OP-REG FROM DATABUS.
5130 :

```

```

5140 :
5150 : * INCREMENT VAX COUNTER :
5160 :
5170 : VAX COUNTER WILL POINT THE 2903 REGISTER ADDRESS SPECIFIED
5180 : BY THE ADDRESSING MODE.
5190 :
5200 INCR      FORM  22X.1B1.89X
5210 :

```

```

5220 : * EXPONENT COUNTER CONTROL :
5230 :
5240 EXPCU     FORM  28X.2B00.82X      : UPCOUNT EXP-COUNTER.
5250 EXPCD     FORM  28X.2B10.82X      : DOWNCOUNT EXP-COUNTER.
5260 :

```

```

5270 :
5280 : * INDEX MODE FLIP FLOP CONTROL :
5290 :
5300 : INDEX MODE FLIPFLOP WILL INDICATE THE INDEX MODE.
5310 :
5320 SINFF     FORM  30X.2B10.80X      : SET INDEX MODE FLIP FLOP.
5330 CINFF     FORM  30X.2B01.80X      : CLEAR INDEX MODE FLIP FLOP.
5340 :

```

```

5350 :
5360 : * CONTINUE FLIP FLOP CONTROL :
5370 :
5380 : A GENERAL FLIP FLOP NAMED CONTINUE WILL WORK FOR
5390 : CONDITION TEST.
5400 :
5410 SCNT      FORM  3X.2B01.107X      : SET CONTINUE FLIP FLOP.
5420 CCNT      FORM  3X.2B10.107X      : CLEAR CONTINUE FLIP FLOP.
5430 :

```


NAVTRAEQUIPCEN 78-C-0157-1

PAGE 16 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE. REV 5.5. 11/02/79

5440 : *****
5450 : * DEFINITION FILE OF SPECIAL ALU FUNCTIONS *
5460 : *****
5470 :

5480 :
5490 : * SPALU,ASRC,ADDR,BSRC,BADR,SPFUNC,DEST,WREN,CYIN,SHSFT,QISHFT
5500 :
5510 SPALU FORM 64X,3VQ0,6VH10,1B0,6VH0,4H0,6VQ60,3VH0,3VQ1,9X,2VQ0,2VQ0,1X
5520 :

5530 :
5540 : * DEFINITION OF SPECIAL FUNCTION FIELD (SPFUNC) :
5550 :
5560 MULTUM EQU 000 ; UNSIGNED MULTIPLICATION.
5570 MULT2C EQU 100 ; 2'S COMPLEMENT MULTIPLICATION.
5580 INCR EQU 200 ; INCREMENT BY 1 OR 2.
5590 CONV EQU 270 ; SIGN MAGNITUDE 1 2'S COMP. CONVERSION.
5600 M2CLC EQU 300 ; 2'S COMPL. MULT. LAST CYCLE.
5610 NORM EQU 400 ; SINGLE LENGTH NORMALIZATION.
5620 NORMAL EQU 500 ; DOUBLE LENGTH NORMALIZATION.
5630 DIV2C EQU 600 ; 2'S COMPL. DIVISION.
5640 D2CC EQU 700 ; 2'S COMPL. DIVISION CORRECTION.
5650 EJECT

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 17 PROJECT MASH-DEFINITION VAX-EMULATOR DEFINITION PHASE, REV 5.5, 11/02/79

5660 END

TOTAL ERRORS

NAVTRAEQUIPCEN 78-C-0157-1

APPENDIX D

VAX-Emulator Instruction Set

VAX EMULATOR ADDRESSING MODE, ASSEMBLY PHASE REV6.0 12/02/79

TITLE VAX EMULATOR ADDRESSING MODE, ASSEMBLY PHASE REV6.0 12/02/79
 OPT T=DFIA,UT:1,L=ADDR.AL:1,N=120
 DRG 0000H

;
;
;
;
;

THIS IS THE DEMONSTRATION OF VAX EMULATOR MICROPROGRAM ASSEMBLY
 PHASE WHICH EMULATES THE WHOLE ADDRESSING MODES AND MOST INSTRU-
 TIONS OF VAX 11/780 COMPUTER. THE PROGRAM CONSIST, THREE PARTITIONS;
 ADDRESSING MODE ROUTINES, FIXED POINT INSTRUCTION SET AND FLOATING
 POINT INSTRUCTION SET.
 EJECT

0110
0120
0130
0140

```

*****
* ADDRESSING MODE SUBROUTINES: SOURCE & DESTINATION *
*****

```

0150
0160
0170
0180
0190
0200
0210
0220

THE ACTUAL OPERANDS WILL BE LOCATED BY ADDRESSING SUBROUTINES WHICH CONSIST SOURCE AND DESTINATION ROUTINES. THE SOURCE ROUTINES MOVE THE CONTENT OF SOURCE OPERAND IN INSTRUCTION FORMAT TO AN INTERNAL SOURCE REGISTER, SRC, INTERNAL TO THE ALU RAM ARRAY. THE DESTINATION ROUTINES MOVE THE CONTENT OF DESTINATION REGISTER, DST, OF ALU RAM ARRAY TO THE LOCATION SPECIFIED BY THE INSTRUCTION FORMAT.

0230
0240
0250
0260
0270
0280
0290
0300

: GENERAL PURPOSE REGISTER ADDRESSING :

0280
0290
0100

MODE 0-3 (LITERAL MODE)

0310
0320
0330
0340
0350
0360
0370

SOURCE SUBROUTINE

```

0330
0310 0000 0011F0204300022ACB104E5E0000 MD03S
0320 0001 000000000A000002F08118C1E0000
0330 0002 0000000000E000000001039DE0010 FLIT
0340 0003 0000000000E000000001039DE0010
0350 0004 0000000000A0000000010FCE0000
0360 0005 000000000A000002F08118C1E0000
0370

```

```

SOURCE SUBROUTINE
ALU,LITERAL,,SRC,ADDR,EXTBL,LWRD,CZERO FLAGS FETCH JMP,FLIT,FLDA
ALU,,,SRC2,ZERO,F,LWRD,CZERO FLAGS RET
ALU,REG,SRC,,SRC,ADD,FLU,LWRD,CZERO,SIO FLAGS CONT
ALU,REG,SRC,,SRC,ADD,FLU,LWRD,CZERO,SIO FLAGS CONT
ALU,CONST,,,SRC,OR,F,LWRD,CZERO FLAGS PIPE,4000H CRET,NBB
ALU,,,SRC2,ZERO,F,LWRD,CZERO FLAGS RET

```

0380
0390
0400
0410

MODE 4 (INDEXED MODE)

0420
0430
0440
0450
0460

SOURCE SUBROUTINE

0470
0480
0490
0500
0510
0520
0530
0540
0550

DESTINATION SUBROUTINE

```
0490 000B 0000000004J00103308000C004000 MD4D
0500 000C 0019F022230000240184C1E0000
0510 000B 0019F020230000270C183C1E0000
0520 000E 0019F020230000280C183C1E0000
0530 000F 0011F0202300002F0C183C1E0000
0540 0010 000000000E0000000000C004000 ADRF
0550
```

```

ALU FLAGS CJMP,AURFAULT,INDEXMOD
ALU,REG,AMODE,,INDEX,ADDR,F,LWRD,CZERO FLAGS SIMFF CJDMO,BYT1 CFET
ALU,REG,INDEX,,INDEX,ADD,F,LWRD,CZERO FLAGS CJDMO,BYT2 CFETCH
ALU,REG,INDEX,,INDEX,ADD,F,LWRD,CZERO FLAGS CJDMO,BYT4 CFETCH
ALU,REG,INDEX,,INDEX,ADD,F,LWRD,CZERO FLAGS JDMO FETCH
ALU FLAGS CONT 1 ADDRESSING MODE ERROR,NOT SPECIFIED

```

0560
0570
0580
0590
0600

MODE 3 (REGISTER DIRECT MODE)

0000

SOURCE SUBROUTINE

0000 0011 000002000A000000*1010AC1E0000 MD55

```

SOURCE SUBROUTINE
ALU,REG,AMODE,,SRC,ADDR,F,LMRD,CZERO,INCR,FLAGS,CRET,NDB

```

NAVTRAEQUIPCEN 78-C-0157-1

AGE 3 ADDRESS MASM VAX EMULATOR ADDRESSING MODE, ASSEMBLY PHASE REV. 0 12/02/79

```

0610 0012 000000000400002F10116C1E0000 ALU,REG,AMODE, SRC2,ADDR,F,LWRD,CZERO FLAGS RET
0620
0630
0640
0650 0013 00000200430013090A206C200000 MD5D DESTINATION SUBROUTINE
0660 0014 000000000E00000004A06C1E0000 ALU,REG,DST1,AMODE,ADDR,F,INS,CZERO FLAGS INCUR JMP,MD5D4,NB8
0670 0015 100000000A00003508000C004000 MD5DA ALU,REG,DST2,AMODE,ADDR,F,LWRD,CZERO FLAGS CONT
0680 0016 0011F0201700000C08000C004000 ALU FLAGS CCNT CRET,CONTIN
0690 0017 000000000A00002F2B116C1E0000 ALU FLAGS FETCH JMP

0700
0710
0720
0730
0740 0017 0100000041002E3310166C1E0000 MD6S MODE 6 ( REGISTER DEFERRED MODE )
0750 MD6SA SOURCE SUBROUTINE
0760 0018 01210010430C18C0F163C000030 ALU,REG,AMODE,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJSR,INXSUB,INDXMOD
0770 0019 002100100A0000092B106C1E0000 ALU,REG,FOUR,EA,ADD,F,NWR,CZERO FLAGS READ,ITYPE LADR
0780 001A 000000000A00002F2B116C1E0000 JMP,MD6SNA,NALIGN
0790 001B 002100100E00000008000C004000 MD6SNA ALU,DIR, SRC,ADDR,F,LWRD,CZERO FLAGS READ,ITYPE CRET,NB8
0800 001C 000000000A0000092B106C1E0000 ALU,DIR, SRC2,ADDR,F,LWRD,CZERO FLAGS RET
0810 001D 012100100E0000000E963C000000 MD6SNA ALU FLAGS READ,ITYPE CONT
0820 001E 002100100E00000008000C004000 ALU,DIR, SRC,ADDR,F,LWRD,CZERO FLAGS CRET,NB8
0830 001F 000000000A00002F2B116C1E0000 ALU,REG,EIGHT,EA,ADD,F,NWR,CZERO FLAGS READ,ITYPE LADR CONT
0840 ALU READ,ITYPE CONT
0850 ALU,DIR, SRC2,ADDR,F,LWRD,CZERO FLAGS RET
0860
0870 0020 0100000041002E3310166C1E0000 MD6D DESTINATION SUBROUTINE
0880 MD6DA ALU,REG,AMODE,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJSR,INXSUB,INDXMOD
0890 0021 01330050430024090A806C000000 MD6DA ALU,REG,EA,FOUR,ADD,F,NWR,CZERO FLAGS WRITE,ITYPE LADR
0900 0022 00800000430024090A806C000000 JMP,MD6DNA,NALIGN
0910 0023 0003F0500E00000008000C004000 ALU,REG,DST2,ADDR,F,NWR,CZERO FLAGS LDOR JMP,MD6DR,NB8
0920 0024 100000000A000003508000C004000 MD6DR ALU FLAGS WRITE,LWORD CONT
0930 0025 0011F0201700000C08000C004000 ALU FLAGS CCNT CRET,CONTIN
0940 0026 000300500E00000008000C004000 MD6DNA ALU FLAGS WRITE,ITYPE CONT
0950 0027 00800000430024090A806C000000 ALU,REG,DST2,ADDR,F,NWR,CZERO FLAGS LDOR JMP,MD6DR,NB8
0960 0028 0103F0500E000003508000C004000 MD6DNA ALU,REG,EA,EIGHT,ADD,F,NWR,CZERO FLAGS WRITE,LWORD LADR CONT
0970 0029 0003F0504300242F08000C004000 ALU FLAGS WRITE,LWORD JMP,MD6DR
0980

0990
1000
1010
1020
1030 002A 000000004300172FAB201C1E4000 MD7S MODE 7 (AUTODECREMENT MODE)
1040
1050
1060
1070 002B 000000004300202FAB201C1E4000 MD7D SOURCE SUBROUTINE
1080 MD7D ALU,DTYPE,AMODE,SUBSR,F,LWRD,CONE FLAGS JUMP,MD6S

1090
1100
1110
1120
1130
1140
1150
1160
1170
1180
1190
1200
1210
1220
1230
1240
1250
1260
1270
1280
1290
1300
1310
1320
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480
1490
1500
1510
1520
1530
1540
1550
1560
1570
1580
1590
1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1700
1710
1720
1730
1740
1750
1760
1770
1780
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030
2040
2050
2060
2070
2080
2090
2100
2110
2120
2130
2140
2150
2160
2170
2180
2190
2200
2210
2220
2230
2240
2250
2260
2270
2280
2290
2300
2310
2320
2330
2340
2350
2360
2370
2380
2390
2400
2410
2420
2430
2440
2450
2460
2470
2480
2490
2500
2510
2520
2530
2540
2550
2560
2570
2580
2590
2600
2610
2620
2630
2640
2650
2660
2670
2680
2690
2700
2710
2720
2730
2740
2750
2760
2770
2780
2790
2800
2810
2820
2830
2840
2850
2860
2870
2880
2890
2900
2910
2920
2930
2940
2950
2960
2970
2980
2990
3000
3010
3020
3030
3040
3050
3060
3070
3080
3090
3100
3110
3120
3130
3140
3150
3160
3170
3180
3190
3200
3210
3220
3230
3240
3250
3260
3270
3280
3290
3300
3310
3320
3330
3340
3350
3360
3370
3380
3390
3400
3410
3420
3430
3440
3450
3460
3470
3480
3490
3500
3510
3520
3530
3540
3550
3560
3570
3580
3590
3600
3610
3620
3630
3640
3650
3660
3670
3680
3690
3700
3710
3720
3730
3740
3750
3760
3770
3780
3790
3800
3810
3820
3830
3840
3850
3860
3870
3880
3890
3900
3910
3920
3930
3940
3950
3960
3970
3980
3990
4000
4010
4020
4030
4040
4050
4060
4070
4080
4090
4100
4110
4120
4130
4140
4150
4160
4170
4180
4190
4200
4210
4220
4230
4240
4250
4260
4270
4280
4290
4300
4310
4320
4330
4340
4350
4360
4370
4380
4390
4400
4410
4420
4430
4440
4450
4460
4470
4480
4490
4500
4510
4520
4530
4540
4550
4560
4570
4580
4590
4600
4610
4620
4630
4640
4650
4660
4670
4680
4690
4700
4710
4720
4730
4740
4750
4760
4770
4780
4790
4800
4810
4820
4830
4840
4850
4860
4870
4880
4890
4900
4910
4920
4930
4940
4950
4960
4970
4980
4990
5000
5010
5020
5030
5040
5050
5060
5070
5080
5090
5100
5110
5120
5130
5140
5150
5160
5170
5180
5190
5200
5210
5220
5230
5240
5250
5260
5270
5280
5290
5300
5310
5320
5330
5340
5350
5360
5370
5380
5390
5400
5410
5420
5430
5440
5450
5460
5470
5480
5490
5500
5510
5520
5530
5540
5550
5560
5570
5580
5590
5600
5610
5620
5630
5640
5650
5660
5670
5680
5690
5700
5710
5720
5730
5740
5750
5760
5770
5780
5790
5800
5810
5820
5830
5840
5850
5860
5870
5880
5890
5900
5910
5920
5930
5940
5950
5960
5970
5980
5990
6000
6010
6020
6030
6040
6050
6060
6070
6080
6090
6100
6110
6120
6130
6140
6150
6160
6170
6180
6190
6200
6210
6220
6230
6240
6250
6260
6270
6280
6290
6300
6310
6320
6330
6340
6350
6360
6370
6380
6390
6400
6410
6420
6430
6440
6450
6460
6470
6480
6490
6500
6510
6520
6530
6540
6550
6560
6570
6580
6590
6600
6610
6620
6630
6640
6650
6660
6670
6680
6690
6700
6710
6720
6730
6740
6750
6760
6770
6780
6790
6800
6810
6820
6830
6840
6850
6860
6870
6880
6890
6900
6910
6920
6930
6940
6950
6960
6970
6980
6990
7000
7010
7020
7030
7040
7050
7060
7070
7080
7090
7100
7110
7120
7130
7140
7150
7160
7170
7180
7190
7200
7210
7220
7230
7240
7250
7260
7270
7280
7290
7300
7310
7320
7330
7340
7350
7360
7370
7380
7390
7400
7410
7420
7430
7440
7450
7460
7470
7480
7490
7500
7510
7520
7530
7540
7550
7560
7570
7580
7590
7600
7610
7620
7630
7640
7650
7660
7670
7680
7690
7700
7710
7720
7730
7740
7750
7760
7770
7780
7790
7800
7810
7820
7830
7840
7850
7860
7870
7880
7890
7900
7910
7920
7930
7940
7950
7960
7970
7980
7990
8000
8010
8020
8030
8040
8050
8060
8070
8080
8090
8100
8110
8120
8130
8140
8150
8160
8170
8180
8190
8200
8210
8220
8230
8240
8250
8260
8270
8280
8290
8300
8310
8320
8330
8340
8350
8360
8370
8380
8390
8400
8410
8420
8430
8440
8450
8460
8470
8480
8490
8500
8510
8520
8530
8540
8550
8560
8570
8580
8590
8600
8610
8620
8630
8640
8650
8660
8670
8680
8690
8700
8710
8720
8730
8740
8750
8760
8770
8780
8790
8800
8810
8820
8830
8840
8850
8860
8870
8880
8890
8900
8910
8920
8930
8940
8950
8960
8970
8980
8990
9000
9010
9020
9030
9040
9050
9060
9070
9080
9090
9100
9110
9120
9130
9140
9150
9160
9170
9180
9190
9200
9210
9220
9230
9240
9250
9260
9270
9280
9290
9300
9310
9320
9330
9340
9350
9360
9370
9380
9390
9400
9410
9420
9430
9440
9450
9460
9470
9480
9490
9500
9510
9520
9530
9540
9550
9560
9570
9580
9590
9600
9610
9620
9630
9640
9650
9660
9670
9680
9690
9700
9710
9720
9730
9740
9750
9760
9770
9780
9790
9800
9810
9820
9830
9840
9850
9860
9870
9880
9890
9900
9910
9920
9930
9940
9950
9960
9970
9980
9990

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 4 ADDRESS MASH VAX EMULATOR ADDRESSING MODE, ASSEMBLY PHASE REV.0 12/02/79

| | | | | |
|------|------|------------------------------|--------|---|
| 1130 | 002C | 0100000041002E3310166C1E0000 | MD8S | ALU,REG,AMODE,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJSB,INXSUB,INDXHL |
| 1140 | 002D | 000000004300182FAB203C1E0000 | | ALU,DTYPE,,AMODE,ADD,F,LWRD,CZERO FLAGS JUMP,MD4SK |
| 1150 | 002E | 010000010A00002F0C163C1E0000 | INXSUB | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF RET |
| 1160 | | | | |
| 1170 | | | | DESTINATION SUBROUTINE |
| 1180 | 002F | 0100000041002E3310166C1E0000 | MD8D | ALU,REG,AMODE,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJSB,INXSUB,INDXMO |
| 1190 | 0030 | 000000004300212FAB203C1E0000 | | ALU,DTYPE,,AMODE,ADD,F,LWRD,CZERO FLAGS JUMP,MD4DK |
| 1200 | | | | |
| 1210 | | | | |
| 1220 | | | | MODE 9 (AUTOINCREMENT DEFERRED MODE) |
| 1230 | | | | |
| 1240 | | | | SOURCE SUBROUTINE |
| 1250 | 0031 | 010000000E00000010004C000000 | MD9S | ALU,REG,AMODE,,ADDR,F,NWR,CZERO FLAGS LADR CONT |
| 1260 | | | | ALU,REG,FOUR,,AMODE,ADD,F,LWRD,CZERO FLAGS READ,LWORD |
| 1270 | 0032 | 0121F010430035080F203C1E0000 | / | LADR CUMP,MD9SNA,NALIGN |
| 1280 | 0033 | 010000004300181328166C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CUMP,MD4SK,NINDXMOD |
| 1290 | 0034 | 010000014300182F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MD4SK |
| 1300 | 0035 | 0021F0100E00000008000C004000 | MD9SNA | ALU,FLAGS,READ,LWORD CONT |
| 1310 | 0036 | 010000004300181328166C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CUMP,MD4SK,NINDXMOD |
| 1320 | 0037 | 010000014300182F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MD4SK |
| 1330 | | | | |
| 1340 | | | | DESTINATION SUBROUTINE |
| 1350 | 0038 | 010000000E00000010004C000000 | MD9D | ALU,REG,AMODE,,ADDR,F,NWR,CZERO FLAGS LADR CONT |
| 1360 | | | | ALU,REG,FOUR,,AMODE,ADD,F,LWRD,CZERO FLAGS LADR |
| 1370 | 0039 | 0100000043003C080F203C1E0000 | / | CUMP,MD9DNA,NALIGN |
| 1380 | 003A | 010000004300211328166C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CUMP,MD4SK,NINDXMOD |
| 1390 | 003B | 010000014300212F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MD4SK |
| 1400 | 003C | 0021F0100E00000008000C004000 | MD9DNA | ALU,READ,LWORD CONT |
| 1410 | 003D | 010000004300211328166C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CUMP,MD4SK,NINDXMOD |
| 1420 | 003E | 010000014300212F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MD4SK |
| 1430 | | | | |
| 1440 | | | | |
| 1450 | | | | MODE A (BYTE DISPLACEMENT) |
| 1460 | | | | |
| 1470 | | | | SOURCE SUBROUTINE |
| 1480 | 003F | 000000000E00000008168C1E0000 | MDAS | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 1490 | 0040 | 0011F0200E0000000816FE3E0000 | | ALU,BSS,,,EA,OR,EXTBL,LWRD,CZERO FLAGS FETCH CONT |
| 1500 | 0041 | 010000004300181310163C1E0000 | | ALU,REG,AMODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CUMP,MD4SK,NINDXMOD |
| 1510 | 0042 | 010000014300182F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MD4SK |
| 1520 | | | | |
| 1530 | | | | DESTINATION SUBROUTINE |
| 1540 | 0043 | 000000000E00000008168C1E0000 | MDAD | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 1550 | 0044 | 0011F0200E0000000816FE3E0000 | | ALU,BSS,,,EA,OR,EXTBL,LWRD,CZERO FLAGS FETCH CONT |
| 1560 | 0045 | 010000004300211310163C1E0000 | | ALU,REG,AMODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CUMP,MD4SK,NINDXMOD |
| 1570 | 0046 | 010000014300212F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MD4SK |
| 1580 | | | | |
| 1590 | | | | |
| 1600 | | | | MODE B (BYTE DISPLACEMENT DEFERRED) |
| 1610 | | | | |
| 1620 | | | | SOURCE SUBROUTINE |
| 1630 | 0047 | 000000000E00000008168C1E0000 | MDBS | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 1640 | 0048 | 0011F0200E0000000816FE3E0000 | | ALU,BSS,,,EA,OR,EXTBL,LWRD,CZERO FLAGS FETCH CONT |

NAVTRAEQUIPCEN 78-C-0157-1

PAUSE 5 ADDRESS MASH VAX EMULATOR ADDRESSING MODE ASSEMBLY PHASE REV6.0 12/02/79

| | | | | |
|------|------|--------------------------------|--------|---|
| 1450 | 0049 | 0100000000E000000010163C1E0000 | | ALU,REG,ANODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CONT |
| 1460 | | | | ALU,REG,FOUR,,EA,ADD,F,LWRD,CZERO FLAGS LADR READ,LWORD |
| 1470 | 004A | 0121F01043004D0B0F143C1E0000 | / | CJMP,MDBSNA,NALIGN |
| 1480 | 004B | 01000000430021132B166C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 1490 | 004C | 0100000143001B2F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 1700 | 004D | 0021F0100E00000000000000000000 | MDBSNA | ALU FLAGS READ,LWORD CONT |
| 1710 | 004E | 0100000043001B132B166C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 1720 | 004F | 0100000143001B2F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 1730 | | | | |
| 1740 | | | | |
| 1750 | | | | |
| 1760 | 0050 | 0000000000E000000008168C1E0000 | MDBD | DESTINATION SUBROUTINE |
| 1770 | 0051 | 0011F0200E00000000816FE5E00000 | | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 1780 | 0052 | 0100000000E000000010163C1E0000 | | ALU,BSD,,EA,OR,EXTBL,LWRD,CZERO FLAGS FETCH CONT |
| 1790 | | | | ALU,REG,ANODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CONT |
| 1800 | 0053 | 0121F0104300540B0F163C1E0000 | / | ALU,REG,FOUR,,EA,ADD,F,LWRD,CZERO FLAGS LADR READ,LWORD |
| 1810 | 0054 | 01000000430021132B166C1E0000 | | CJMP,MDBDNA,NALIGN |
| 1820 | 0055 | 010000014300212F0C163C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 1830 | 0056 | 0021F0100E00000000800000004000 | MDBDNA | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 1840 | 0057 | 01000000430021132B166C1E0000 | | ALU FLAGS READ,LWORD CONT |
| 1850 | 0058 | 010000014300212F0C163C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 1860 | | | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 1870 | | | | |
| 1880 | | | | |
| 1890 | | | | |
| 1900 | | | | |
| 1910 | 0059 | 0000000000E000000008168C1E0000 | MDCS | MODE C (WORD DISPLACEMENT) |
| 1920 | 005A | 0011F0200E00000000816FC0200000 | | SOURCE SUBROUTINE |
| 1930 | 005B | 0011F0200E00000000816FEDC00000 | | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 1940 | 005C | 0100000043001B1310163C1E0000 | | ALU,BSD,,EA,OR,EXTBL,LWRD,CZERO FLAGS FETCH CONT |
| 1950 | 005D | 0100000143001B2F0C163C1E0000 | | ALU,BSD,,EA,OR,EXTBL,B321,CZERO FLAGS FETCH CONT |
| 1960 | | | | ALU,REG,ANODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 1970 | | | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 1980 | | | | |
| 1990 | 005E | 0000000000E000000008168C1E0000 | MDCD | DESTINATION SUBROUTINE |
| 2000 | 005F | 0011F0200E00000000816FC0200000 | | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 2010 | 0060 | 0011F0200E00000000816FEDC00000 | | ALU,BSD,,EA,OR,EXTBL,B321,CZERO FLAGS FETCH CONT |
| 2020 | 0061 | 010000004300211310163C1E0000 | | ALU,REG,ANODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 2030 | 0062 | 010000014300212F0C163C1E0000 | | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 2040 | | | | |
| 2050 | | | | |
| 2060 | | | | |
| 2070 | | | | |
| 2080 | | | | |
| 2090 | 0063 | 0000000000E000000008168C1E0000 | MDEE | MODE D (WORD DISPLACEMENT DEFERRED) |
| 2100 | 0064 | 0011F0200E00000000816FC0200000 | | SOURCE SUBROUTINE |
| 2110 | 0065 | 0011F0200E00000000816FEDC00000 | | ALU,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 2120 | 0066 | 0100000000E000000010163C1E0000 | | ALU,BSD,,EA,OR,EXTBL,B321,CZERO FLAGS FETCH CONT |
| 2130 | | | | ALU,REG,ANODE,,EA,ADD,F,LWRD,CZERO FLAGS LADR CONT |
| 2140 | 0067 | 0121F01043004A0B0F143C1E0000 | | ALU,REG,FOUR,,EA,ADD,F,LWRD,CZERO FLAGS READ,LWORD LADR |
| 2150 | 0068 | 0100000043001B132B166C1E0000 | | CJMP,MDBSNA,NALIGN |
| 2160 | 0069 | 0100000143001B2F0C163C1E0000 | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |
| 2170 | 006A | 0021F0100E00000000800000004000 | MDBSNA | ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS LADR CINFF JUMP,MDBSK |
| 2180 | 006B | 0100000043001B132B166C1E0000 | | ALU FLAGS READ,LWORD CONT |
| | | | | ALU,DIR,,EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MDBSK,NINDXMOD |

NAVTRAEQUIPCEN 78-C-0157-1

PAUL A ADDRESS MASH VAX EMULATOR ADDRESSING MODE ASSEMBLY PHASE REV6.0 12-02/79

```

2190 006C 01000001430012F0C1A3C1E0000
2200
2210
2220
2230
2240 006D 000000000E000000001A8C1E0000
2250 006E 0011F0200E000000A81AF0C020000
2260 006F 0011F0200E000000A81AF0C00000
2270 0070 01000000E000000101A3C1E0000
2280
2290 0071 0121F0104300740B0F1A3C1E0000
2300 0072 01000000430021132B1A6C1E0000
2310 0073 010000014300212F0C1A3C1E0000
2320 0074 0021F0100E000000000000C004000
2330 0075 01000000430021132B1A6C1E0000
2340 0076 010000014300212F0C1A3C1E0000
2350
2360
2370
2380
2390 0077 00000000E000000001A8C1E0000
2400 0078 0011F0200E000000A81AF0C1E0000
2410 0079 0011F0200E000000A81AF0C020000
2420 007A 0011F0200E000000A81AF0C040000
2430 007B 0011F0200E000000A81AF0C080000
2440 007C 0011F0200E000000A81AF0C000000
2450 007D 0100000043001B13101A3C1E0000
2460 007E 01000001430012F0C1A3C1E0000
2470
2480
2490
2500 007F 00000000E000000001A8C1E0000
2510 0080 0011F0200E000000A81AF0C020000
2520 0081 0011F0200E000000A81AF0C040000
2530 0082 0011F0200E000000A81AF0C080000
2540 0083 0011F0200E000000A81AF0C100000
2550 0084 0100000043002113101A3C1E0000
2560 0085 010000014300212F0C1A3C1E0000
2570
2580
2590
2600
2610
2620 0086 00000000E000000001A8C1E0000
2630 0087 0011F0200E000000A81AF0C020000
2640 0088 0011F0200E000000A81AF0C040000
2650 0089 0011F0200E000000A81AF0C080000
2660 008A 0011F0200E000000A81AF0C100000
2670 008B 01000000E000000101A3C1E0000
2680
2690 008C 0121F0104300B0B0F1A3C1E0000
2700 008D 0100000043001B132B1A6C1E0000
2710 008E 000000014300212F0C1A3C1E0000
2720 008F 0021F0100E0000000000C004000
2730
2740
2750
2760
2770
2780
2790
2800
2810
2820
2830
2840
2850
2860
2870
2880
2890
2900
2910
2920
2930
2940
2950
2960
2970
2980
2990
3000
3010
3020
3030
3040
3050
3060
3070
3080
3090
3100
3110
3120
3130
3140
3150
3160
3170
3180
3190
3200
3210
3220
3230
3240
3250
3260
3270
3280
3290
3300
3310
3320
3330
3340
3350
3360
3370
3380
3390
3400
3410
3420
3430
3440
3450
3460
3470
3480
3490
3500
3510
3520
3530
3540
3550
3560
3570
3580
3590
3600
3610
3620
3630
3640
3650
3660
3670
3680
3690
3700
3710
3720
3730
3740
3750
3760
3770
3780
3790
3800
3810
3820
3830
3840
3850
3860
3870
3880
3890
3900
3910
3920
3930
3940
3950
3960
3970
3980
3990
4000
4010
4020
4030
4040
4050
4060
4070
4080
4090
4100
4110
4120
4130
4140
4150
4160
4170
4180
4190
4200
4210
4220
4230
4240
4250
4260
4270
4280
4290
4300
4310
4320
4330
4340
4350
4360
4370
4380
4390
4400
4410
4420
4430
4440
4450
4460
4470
4480
4490
4500
4510
4520
4530
4540
4550
4560
4570
4580
4590
4600
4610
4620
4630
4640
4650
4660
4670
4680
4690
4700
4710
4720
4730
4740
4750
4760
4770
4780
4790
4800
4810
4820
4830
4840
4850
4860
4870
4880
4890
4900
4910
4920
4930
4940
4950
4960
4970
4980
4990
5000
5010
5020
5030
5040
5050
5060
5070
5080
5090
5100
5110
5120
5130
5140
5150
5160
5170
5180
5190
5200
5210
5220
5230
5240
5250
5260
5270
5280
5290
5300
5310
5320
5330
5340
5350
5360
5370
5380
5390
5400
5410
5420
5430
5440
5450
5460
5470
5480
5490
5500
5510
5520
5530
5540
5550
5560
5570
5580
5590
5600
5610
5620
5630
5640
5650
5660
5670
5680
5690
5700
5710
5720
5730
5740
5750
5760
5770
5780
5790
5800
5810
5820
5830
5840
5850
5860
5870
5880
5890
5900
5910
5920
5930
5940
5950
5960
5970
5980
5990
6000
6010
6020
6030
6040
6050
6060
6070
6080
6090
6100
6110
6120
6130
6140
6150
6160
6170
6180
6190
6200
6210
6220
6230
6240
6250
6260
6270
6280
6290
6300
6310
6320
6330
6340
6350
6360
6370
6380
6390
6400
6410
6420
6430
6440
6450
6460
6470
6480
6490
6500
6510
6520
6530
6540
6550
6560
6570
6580
6590
6600
6610
6620
6630
6640
6650
6660
6670
6680
6690
6700
6710
6720
6730
6740
6750
6760
6770
6780
6790
6800
6810
6820
6830
6840
6850
6860
6870
6880
6890
6900
6910
6920
6930
6940
6950
6960
6970
6980
6990
7000
7010
7020
7030
7040
7050
7060
7070
7080
7090
7100
7110
7120
7130
7140
7150
7160
7170
7180
7190
7200
7210
7220
7230
7240
7250
7260
7270
7280
7290
7300
7310
7320
7330
7340
7350
7360
7370
7380
7390
7400
7410
7420
7430
7440
7450
7460
7470
7480
7490
7500
7510
7520
7530
7540
7550
7560
7570
7580
7590
7600
7610
7620
7630
7640
7650
7660
7670
7680
7690
7700
7710
7720
7730
7740
7750
7760
7770
7780
7790
7800
7810
7820
7830
7840
7850
7860
7870
7880
7890
7900
7910
7920
7930
794
```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 7 ADDRESS MASH VAX EMULATOR ADDRESSING MODE ASSEMBLY PHASE REV4.0 12/02/79

| | | | |
|------|------|-------------------------------|---|
| 2730 | 0090 | 010000004300181328166C1E0000 | ALU,DIR,..EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MD6SK,NINDXMOD |
| 2740 | 0091 | 000000004300182F0C163C1E0000 | ALU,REG,INDEX,..EA,ADD,F,LWRD,CZERO FLAGS JUMP,MD6SK |
| 2750 | | | |
| 2760 | | | |
| 2770 | | | |
| 2780 | 0092 | 000000000E00000008168C1E0000 | MDFD DESTINATION SUBROUTINE |
| 2790 | 0093 | 0011F0200E00000004816FC020000 | ALU,...EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 2800 | 0094 | 0011F0200E00000004816FC040000 | ALU,BSS,..EA,OR,F,B0,CZERO FLAGS FETCH CONT |
| 2810 | 0095 | 0011F0200E00000004816FC080000 | ALU,BSS,..EA,OR,F,B1,CZERO FLAGS FETCH CONT |
| 2820 | 0096 | 0011F0200E00000004816FC100000 | ALU,BSS,..EA,OR,F,B2,CZERO FLAGS FETCH CONT |
| 2830 | 0097 | 010000000E000000010163C1E0000 | ALU,BSS,..EA,OR,F,B3,CZERO FLAGS FETCH CONT |
| 2840 | | | ALU,REG,AMODE,..EA,ADD,F,LWRD,CZERO FLAGS LADR CONT |
| 2850 | 0098 | 0121F01043009B080F163C1E0000 | ALU,REG,FOUR,..EA,ADD,F,LWRD,CZERO FLAGS LADR READ,LWORD |
| 2860 | 0099 | 010000004300211328166C1E0000 | CJMP,MDFDNA,NALIGN |
| 2870 | 009A | 000000014300212F0C163C1E0000 | ALU,DIR,..EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MD6DA,NINDXMOD |
| 2880 | 009B | 0021F0100E00000008000C004000 | ALU,REG,INDEX,..EA,ADD,F,LWRD,CZERO FLAGS CINTF JUMP,MD6DK |
| 2890 | 009C | 010000004300211328166C1E0000 | ALU FLAGS READ,LWORD CONT |
| 2900 | 009D | 010000004300212F0C163C1E3C00 | ALU,DIR,..EA,ADDR,F,LWRD,CZERO FLAGS LADR CJMP,MD6DK,NINDXMOD |
| 2910 | | | ALU,REG,INDEX,..EA,ADD,F,LWRD,CZERO FLAGS ALL LADR JUMP,MD6DA |

2920
2930
2940
2950
2960
2970
2980
2990

PROGRAM COUNTER ADDRESSING

MODE 8 (IMMEDIATE MODE)

| | | | |
|------|------|-------------------------------|---|
| 3000 | 009E | 000000004100A833E8108C1E0000 | PCMD8S SOURCE SUBROUTINE |
| 3010 | 009F | 0011F0200A00000266810FC020000 | ALU,NOSRC,..SRC,ZERO,F,LWRD,CZERO FLAGS CJBB,IMHX,INDXMOD |
| 3020 | 00A0 | 0011F0200A00000276810FC040000 | ALU,BSS,..SRC,OR,F,B0,CZERO FLAGS FETCH CRET,BYT1 |
| 3030 | 00A1 | 0011F0200E00000004810FC080000 | ALU,BSS,..SRC,OR,F,B1,CZERO FLAGS FETCH CRET,BYT2 |
| 3040 | 00A2 | 0011F0200A00000286810FC100000 | ALU,BSS,..SRC,OR,F,B2,CZERO FLAGS FETCH CONT |
| 3050 | 00A3 | 000000000E0000000E8110C1E0000 | ALU,BSS,..SRC,OR,F,B3,CZERO FLAGS FETCH CRET,BYT4 |
| 3060 | 00A4 | 0011F0200E00000004811FC020000 | ALU,NOSRC,..SRC2,ZERO,F,LWRD,CZERO FLAGS CONT |
| 3070 | 00A5 | 0011F0200E00000004811FC040000 | ALU,BSS,..SRC2,OR,F,B0,CZERO FLAGS FETCH CONT |
| 3080 | 00A6 | 0011F0200E00000004811FC080000 | ALU,BSS,..SRC2,OR,F,B1,CZERO FLAGS FETCH CONT |
| 3090 | 00A7 | 0011F0200E00000004811FC100000 | ALU,BSS,..SRC2,OR,F,B2,CZERO FLAGS FETCH CONT |
| 3100 | 00A8 | 000000010A000002F08000C004000 | ALU,BSS,..SRC2,OR,F,B3,CZERO FLAGS FETCH CONT |
| 3110 | | | ALU FLAGS CINTF RET |

MODE 9 (ABSOLUTE MODE)

| | | | |
|------|------|-------------------------------|--|
| 3120 | | | |
| 3130 | | | |
| 3140 | | | |
| 3150 | | | |
| 3160 | 00A9 | 000000000E0000000E8168C1E0000 | PCMD9S SOURCE SUBROUTINE |
| 3170 | 00AA | 0011F0200E00000004816FC020000 | ALU,NOSRC,..EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 3180 | 00AB | 0011F0200E00000004816FC040000 | ALU,BSS,..EA,OR,F,B0,CZERO FLAGS FETCH CONT |
| 3190 | 00AC | 0011F0200E00000004816FC080000 | ALU,BSS,..EA,OR,F,B1,CZERO FLAGS FETCH CONT |
| 3200 | 00AD | 0011F0204300AF136816FC100000 | ALU,BSS,..EA,OR,F,B2,CZERO FLAGS FETCH CONT |
| 3210 | 00AE | 000000010E0000000C163C1E3C00 | ALU,BSS,..EA,OR,F,B3,CZERO FLAGS FETCH CJMP,PC9SK,NINDXMOD |
| 3220 | 00AF | 010000004300182F0B000C0000000 | ALU,REG,INDEX,..EA,ADD,F,LWRD,CZERO FLAGS ALL CINTF CONT |
| 3230 | | | ALU,REG,EA,..ADDR,F,NWR,CZERO FLAGS LADR JUMP,MD6SK |

3240

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 8 ADDRESS MASH VAX EMULATOR ADDRESSING MODE: ASSEMBLY PHASE REV. 0 12/02/79

```

3250
3260 0080 00000000000000000000000000000000 PCND9D DESTINATION SUBROUTINE
3270 0081 0011F0200E000000000000000000000000 ALU.NOSRC...EA.ZERO.F.LWRD.CZERO FLAGS CONT
3280 0082 0011F0200E000000000000000000000000 ALU.BSB...EA.OR.F.B0.CZERO FLAGS FETCH CONT
3290 0083 0011F0200E000000000000000000000000 ALU.BSB...EA.OR.F.B1.CZERO FLAGS FETCH CONT
3300 0084 0011F02043000000000000000000000000 ALU.BSB...EA.OR.F.B2.CZERO FLAGS FETCH CONT
3310 0085 0000000010E00000000000000000000000 ALU.BSB...EA.UR.F.B3.CZERO FLAGS FETCH CUMP.PC9DK.NINBND
3320 0086 010000004300212F0000000000000000 PC9DK ALU.REG.INDEX..EA.ADD.F.LWRD.CZERO FLAGS ALL CINF CONT
3330 ALU.REG.EA...ADDR.F.NWR.CZERO FLAGS LADR JUMP.MD6DK

3340
3350
3360
3370
3380 0087 0000000000E000000000000000000000 PCNDAS SOURCE SUBROUTINE
3390 0088 0011F0200E000000000000000000000000 ALU.NOSRC...EA.ZERO.F.LWRD.CZERO FLAGS CONT
3400 0089 000000004300181307943C1E3C00 ALU.BSB...EA.OR.EXTBL.LWRD.CZERO FLAGS FETCH CONT
3410 008A 010000014300182F0C143C1E3C00 ALU.REG.PC...EA.ADD.F.LWRD.CZERO FLAGS CUMP.MD6A.NINBND
3420 ALU.REG.INDEX..EA.ADD.F.LWRD.CZERO FLAGS ALL CINF LADR JUMP.MD6A

3430
3440
3450 008B 0000000000E000000000000000000000 PCNDAD DESTINATION SUBROUTINE
3460 008C 0011F0200E000000000000000000000000 ALU.NOSRC...EA.ZERO.F.LWRD.CZERO FLAGS CONT
3470 008D 010000004300211307943C1E3C00 ALU.BSB...EA.OR.EXTBL.LWRD.CZERO FLAGS FETCH CONT
3480 008E 010000014300212F0C143C1E3C00 ALU.REG.PC...EA.ADD.F.LWRD.CZERO FLAGS LADR CUMP.MD6DK.NINBND
3490 ALU.REG.INDEX..EA.ADD.F.LWRD.CZERO FLAGS ALL CINF LADR JUMP.MD6DK

3500
3510
3520
3530
3540 008F 0000000000E000000000000000000000 PCND8S SOURCE SUBROUTINE
3550 0090 0011F0200E000000000000000000000000 ALU.NOSRC...EA.ZERO.F.LWRD.CZERO FLAGS CONT
3560 0091 0100000000E00000000000000000000000 ALU.BSB...EA.OR.EXTBL.LWRD.CZERO FLAGS FETCH CONT
3570 ALU.REG.PC...EA.ADD.F.LWRD.CZERO FLAGS ALL LADR CONT
3580 0092 0121F01043004D080F143C1E3C00 ALU.REG.FOUR..EA.ADD.F.LWRD.CZERO FLAGS ALL LADR READ.LWORD
3590 0093 010000004300181328144C1E0000 CUMP.MD6A.NALIN
3600 0094 010000004300182F0C143C1E3C00 ALU.DIR...EA.ADD.F.LWRD.CZERO FLAGS LADR CUMP.MD6A.NINBND
3610 ALU.REG.INDEX..EA.ADD.F.LWRD.CZERO FLAGS ALL LADR JUMP.MD6A

3620
3630
3640 0095 0000000000E000000000000000000000 PCND8D DESTINATION SUBROUTINE
3650 0096 0011F0200E000000000000000000000000 ALU.NOSRC...EA.ZERO.F.LWRD.CZERO FLAGS CONT
3660 0097 0100000000E00000000000000000000000 ALU.BSB...EA.OR.EXTBL.LWRD.CZERO FLAGS FETCH CONT
3670 ALU.REG.PC...EA.ADD.F.LWRD.CZERO FLAGS ALL LADR CONT
3680 0098 0121F010430054080F143C1E3C00 ALU.REG.FOUR..EA.ADD.F.LWRD.CZERO FLAGS ALL LADR READ.LWORD
3690 0099 010000004300211328144C1E0000 CUMP.MD6A.NALIN
3700 009A 010000014300212F0C143C1E3C00 ALU.DIR...EA.ADD.F.LWRD.CZERO FLAGS LADR CUMP.MD6A.NINBND
3710 ALU.REG.INDEX..EA.ADD.F.LWRD.CZERO FLAGS ALL LADR CINF JUMP.MD6A

3720
3730
3740
3750
3760 009B 0000000000E000000000000000000000 PCND8C SOURCE SUBROUTINE
3770 ALU.NOSRC...EA.ZERO.F.LWRD.CZERO FLAGS CONT

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 9 ADDRESS MASH VAX EMULATOR ADDRESSING MODE, ASSEMBLY PHASE REV. 0 12/02/79

| | | | |
|------|------|------------------------------|--|
| 3770 | 00CC | 0011F0200E0000006816FC020000 | ALU,BSB,,,EA,OR,F,80,CZERO FLAGS FETCH CONT |
| 3780 | 00CD | 0011F0200E0000006816FEDC0000 | ALU,BSB,,,EA,OR,EXTW,B321,CZERO FLAGS FETCH CONT |
| 3790 | 00CE | 010000004300181307963C1E3C00 | ALU,REG,PC,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CJMP,MD6SK,NINDXMC |
| 3800 | 00CF | 010000014300182F0C163C1E3C00 | ALU,REG,INDEX,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CINTF JUMP,MD6S |
| 3810 | | | |
| 3820 | | | |
| 3830 | | | |
| 3840 | 00D0 | 000000000E000000E8168C1E0000 | PCMDCD DESTINATION SUBROUTINE |
| 3850 | 00D1 | 0011F0200E0000006816FC020000 | ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 3860 | 00D2 | 0011F0200E0000006816FEDC0000 | ALU,BSB,,,EA,OR,F,80,CZERO FLAGS FETCH CONT |
| 3870 | 00D3 | 010000004300211307963C1E3C00 | ALU,BSB,,,EA,OR,EXTW,B321,CZERO FLAGS FETCH CONT |
| 3880 | 00D4 | 010000014300212F0C163C1E3C00 | ALU,REG,PC,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CJMP,MD6SK,NINDXMC |
| 3890 | | | ALU,REG,INDEX,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CINTF JUMP,MD6S |
| 3900 | | | |
| 3910 | | | |
| 3920 | | | |
| 3930 | | | |
| 3940 | 00D5 | 000000000E000000E8168C1E0000 | PCMDDS SOURCE SUBROUTINE |
| 3950 | 00D6 | 0011F0200E0000006816FC020000 | ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 3960 | 00D7 | 0011F0200E0000006816FEDC0000 | ALU,BSB,,,EA,OR,F,80,CZERO FLAGS FETCH CONT |
| 3970 | 00D8 | 010000000E00000007963C1E3C00 | ALU,BSB,,,EA,OR,EXTW,B321,CZERO FLAGS FETCH CONT |
| 3980 | | | ALU,REG,PC,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CONT |
| 3990 | 00D9 | 0121F01043004A0B0F163C1E3C00 | ALU,REG,FOUR,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL READ,LWORD LADR |
| 4000 | 00DA | 010000004300181328166C1E0000 | CJMP,MDDNA,NALIGN |
| 4010 | 00DB | 010000004300182F0C163C1E3C00 | ALU,DIR,,,EA,ADD,F,LWRD,CZERO FLAGS LADR CJMP,MD6SK,NINDXMOD |
| 4020 | | | ALU,REG,INDEX,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JUMP,MD6SK |
| 4030 | | | |
| 4040 | | | |
| 4050 | 00DC | 000000000E000000E8168C1E0000 | PCMDDD DESTINATION SUBROUTINE |
| 4060 | 00DD | 0011F0200E0000006816FC020000 | ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 4070 | 00DE | 0011F0200E0000006816FEDC0000 | ALU,BSB,,,EA,OR,F,80,CZERO FLAGS FETCH CONT |
| 4080 | 00DF | 010000000E00000007963C1E3C00 | ALU,BSB,,,EA,OR,EXTW,B321,CZERO FLAGS FETCH CONT |
| 4090 | | | ALU,REG,PC,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CONT |
| 4100 | 00E0 | 0121F0104300740B0F163C1E3C00 | ALU,REG,FOUR,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL READ,LWORD LADR |
| 4110 | 00E1 | 000000004300211328164C1E0000 | CJMP,MDDNA,NALIGN |
| 4120 | 00E2 | 010000004300212F0C163C1E3C00 | ALU,DIR,,,EA,ADD,F,LWRD,CZERO FLAGS CJMP,MD6SK,NINDXMOD |
| 4130 | | | ALU,REG,INDEX,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JUMP,MD6SK |
| 4140 | | | |
| 4150 | | | |
| 4160 | | | |
| 4170 | | | |
| 4180 | 00E3 | 000000000E000000E8168C1E0000 | PCMDDE SOURCE SUBROUTINE |
| 4190 | 00E4 | 0011F0200E0000006816FC020000 | ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAOS CONT |
| 4200 | 00E5 | 0011F0200E0000006816FC040000 | ALU,BSB,,,EA,OR,F,80,CZERO FLAGS FETCH CONT |
| 4210 | 00E6 | 0011F0200E0000006816FC080000 | ALU,BSB,,,EA,OR,F,81,CZERO FLAGS FETCH CONT |
| 4220 | 00E7 | 0011F0200E0000006816FC100000 | ALU,BSB,,,EA,OR,F,82,CZERO FLAGS FETCH CONT |
| 4230 | 00E8 | 010000004300181307963C1E3C00 | ALU,BSB,,,EA,OR,F,83,CZERO FLAGS FETCH CONT |
| 4240 | 00E9 | 010000004300182F0C163C1E3C00 | ALU,REG,PC,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CJMP,MD6SK,NINDXMC |
| 4250 | | | ALU,REG,INDEX,,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JUMP,MD6SK |
| 4260 | | | |
| 4270 | | | |
| 4280 | 00EA | 000000000E000000E8168C1E0000 | PCMDDE DESTINATION SUBROUTINE |
| 4290 | 00EB | 0011F0200E0000006816FC020000 | ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT |
| 4300 | 00EC | 0011F0200E0000006816FC040000 | ALU,BSB,,,EA,OR,F,80,CZERO FLAGS FETCH CONT |

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 10 ADDRESS MASH VAX EMULATOR ADDRESSING MODE, ASSEMBLY PHASE REV. 0 12/02/79

```

4310 00ED 0011F0200E0000004816FC080000 ALU,BSB,,,EA,OR,F,B2,CZERO FLAGS FETCH CONT
4320 00EE 0011F0200E0000004816FC100000 ALU,BSB,,,EA,OR,F,B3,CZERO FLAGS FETCH CONT
4330 00EF 010000004300211307963C1E3C00 ALU,REG,PC,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JMP,MD&DR,NINDXMOD
4340 00F0 010000004300212F0C163C1E3C00 ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JMP,MD&DR
4350

4360
4370
4380
4390
4400 00F1 000000000E000000E8168C1E0000 PCNDFS SOURCE SUBROUTINE
4410 00F2 0011F0200E0000004816FC020000 ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT
4420 00F3 0011F0200E0000004816FC040000 ALU,BSB,,,EA,OR,F,B0,CZERO FLAGS FETCH CONT
4430 00F4 0011F0200E0000004816FC080000 ALU,BSB,,,EA,OR,F,B1,CZERO FLAGS FETCH CONT
4440 00F5 0011F0200E0000004816FC100000 ALU,BSB,,,EA,OR,F,B2,CZERO FLAGS FETCH CONT
4450 00F6 010000000E00000007963C1E3C00 ALU,BSB,,,EA,OR,F,B3,CZERO FLAGS FETCH CONT
4460 ALU,REG,PC,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CONT
4470 00F7 0121F01043008F080F163C1E3C00 ALU,REG,FOUR,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR READ,LWORD
4480 00F8 010000004300181328166C1E0000 CUMP,MDFSNA,NALIGN
4490 00F9 010000004300162F0C163C1E3C00 ALU,DIR,,,EA,ADDR,F,LWRD,CZERO FLAGS LADR JMP,MD&DR,NINDXMOD
4500 ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JMP,MD&DR
4510
4520
4530 00FA 000000000E000000E8168C1E0000 DESTINATION SUBROUTINE
4540 00FB 0011F0200E0000004816FC020000 ALU,NOSRC,,,EA,ZERO,F,LWRD,CZERO FLAGS CONT
4550 00FC 0011F0200E0000004816FC040000 ALU,BSB,,,EA,OR,F,B0,CZERO FLAGS FETCH CONT
4560 00FD 0011F0200E0000004816FC080000 ALU,BSB,,,EA,OR,F,B1,CZERO FLAGS FETCH CONT
4570 00FE 0011F0200E0000004816FC100000 ALU,BSB,,,EA,OR,F,B2,CZERO FLAGS FETCH CONT
4580 00FF 010000000E00000007963C1E3C00 ALU,BSB,,,EA,OR,F,B3,CZERO FLAGS FETCH CONT
4590 ALU,REG,PC,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR CONT
4600 0100 0121F010430098080F163C1E3C00 ALU,REG,FOUR,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR READ,LWORD
4610 0101 010000004300212F28166C1E0000 CUMP,MDFSNA,NALIGN
4620 0102 010000004300211328166C1E0000 ALU,DIR,,,EA,ADDR,F,LWRD,CZERO FLAGS LADR JMP,MD&DR,NINDXMOD
4630 0103 010000004300212F0C163C1E3C00 ALU,REG,INDEX,,EA,ADD,F,LWRD,CZERO FLAGS,ALL LADR JMP,MD&DR
4640 EJECT

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 1 FIXED MASH VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV4.0
*** INVALID

0010
0020
0030
0040

TITLE VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV4.0
OPT T=DFTA.DT:1,L=FIXED.AL:1,N=120
ORG 0000H

0050
0060
0070

;
; INTEGER INSTRUCTION SET CONSISTS GENERAL ARITHMETIC, LOGIC
; BRANCHING AND SOME SPECIAL EDITING INSTRUCTIONS.
EJECT

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 2 FIXED MASM VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV.0

```

0080      ; *****
0090      ; * MICROROUTINES OF INTEGER INSTRUCTIONS *
0100      ; *****
0110

0120      ;
0130      ; MOVE
0140      ; FORMAT: MOV SRC,RX,DST,WX
0150      ;
0160 0000 0011F0202100004F08000C004000 MOV ALU FLAGS JSND FETCH
0170 0001 0099F0202300000908144C1E1C00 ALU,REG, SRC, DST, ADDR, F, LWRD, CZERO FLAGS, UNZ LDOR CJND, M00 CFETCH
0180 0002 0091F0202300002F08954C1E1C00 ALU,REG, SRC2, DST2, ADDR, F, LWRD, CZERO FLAGS, UNZ LDOR JDND FETCH
0190

0200      ;
0210      ; MOVE ZERO-EXTENDED
0220      ; FORMAT: MOVZ SRC,RX, DST,WY
0230      ;
0240 0003 0011F0202100004FEB148C1E0000 MOVZBL ALU, NOSRC, DST, ZERO, F, LWRD, CZERO FLAGS JSND FETCH
0250 0004 0011F0202300002F0814FC021C00 ALU, REG, SRC, DST, OR, F, BO, CZERO FLAGS, UNZ JDND FETCH
0260      ;
0270 0005 0011F0202100004FEB148C1E0000 MOVZWL ALU, NOSRC, DST, ZERO, F, LWRD, CZERO FLAGS JSND FETCH
0280 0006 0011F0202300002F0814FC041C00 ALU, REG, SRC, DST, OR, F, WRD, CZERO FLAGS, UNZ JDND FETCH
0290

0300      ;
0310      ; MOVE NEGATED
0320      ; FORMAT: OPCODE SRC,RX,DST,WX
0330      ;
0340 0007 0011F0202100004FEB148C1E0000 HNEGL ALU, NOSRC, DST, ZERO, F, LWRD, CZERO FLAGS JSND FETCH
0350 0008 0011F0202300002F08141C1E7C00 ALU, REG, SRC, DST, SUBSR, F, LWRD, COME FLAGS, ALL JDND FETCH
0360

0370      ;
0380      ; MOVE COMPLEMENTED
0390      ; FORMAT: OPCODE SRC,RX,DST,WX
0400      ;
0410 0009 0011F0202100004F08000C004000 HCONL ALU FLAGS JSND FETCH
0420 000A 0011F0202300002F08147C1E1C00 ALU, REG, SRC, DST, ADDR, F, LWRD, CZERO FLAGS, UNZ JDND FETCH
0430

0440      ;
0450      ; PUSH LONGWORD
0460      ; FORMAT: OPCODE SRC,RL
0470      ;
0480 000B 000000002100004F08000C004000 PUSHL ALU FLAGS JSND
0490 000C 010000000E0000000F0E1C1E0000 ALU, REG, FOUR, SP, SUBSR, F, LWRD, CZERO FLAGS LADR CONT
0500 000D 0003F0800E00000008004C003C00 ALU, REG, SRC, ADDR, F, HWR, CZERO FLAGS, ALL WRITE, LWORD CONT
0510 000E 0011F0201700000C08000C004000 ALU FLAGS FETCH JMAP
0520

0530      ;

```

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 3 FIXED MASM VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV.0

```

0540      ; INCREMENT
0550      ;
0560      ;
0570 000F 000000002100004F08000C004000 INC ALU FLAGS JSND
0580 0010 0091F0202300002F08146C203C00 ALU,REG,SRC,,DST,ADDR,F,INS,CONE FLAGS,UNZ LDOR JDND FETCH
0590      ;

0600      ;
0610      ; DECREMENT
0620      ;
0630 0011 000000002100004F08146C1E0000 DEC ALU,REG,,DST,ZERO,F,LWRD,CZERO FLAGS JSND
0640 0012 0091F0202300002F08142C1E3C00 ALU,REG,SRC,,DST,SUB,F,LWRD,CZERO FLAGS,ALL LDOR JDND FETCH
0650      ;

0660      ;
0670      ; CLEAR
0680      ;
0690      ;
0700 0013 0099F02023000009E8148C201C00 CLR ALU,NOSRC,,DST,ZERO,F,INS,CZERO FLAGS,UNZ LDOR CJND,NBS CFET
0710 0014 0091F0202300002F08158C201C00 ALU,NOSRC,,DST2,ZERO,F,INS,CZERO FLAGS,UNZ LDOR JDND FETCH
0720      ;

0730      ;
0740      ; COMPARE
0750      ;
0760      ;
0770 0015 0011F0202100004F08000C004000 CNPL ALU FLAGS,NONE JSND FETCH
0780 0016 0011F0202100004F08146C1E0000 ALU,REG,SRC,,DST,ADDR,F,LWRD,CZERO FLAGS JSND FETCH
0790 0017 0011F0201700000C08142C1E3C00 ALU,REG,SRC,,DST,SUB,F,LWRD,CZERO FLAGS,ALL JNAP FETCH
0800      ;

0810      ;
0820      ; CONVERT
0830      ;
0840      ;
0850 0018 0011F0202100004F08000C004000 CVTBM ALU FLAGS JSND FETCH
0860 0019 0011F0202300002F08146F463C00 ALU,REG,SRC,,DST,ADDR,EXTB,WORD,CZERO FLAGS,ALL JDND FETCH
0870      ;
0880 001A 0011F0202100004F08000C004000 CVTBL ALU FLAGS JSND FETCH
0890 001B 0011F0202300002F08146E5E3C00 ALU,REG,SRC,,DST,ADDR,EXTBL,LWRD,CZERO FLAGS,ALL JDND FETCH
0900      ;

0910      ;
0920      ; TEST
0930      ;
0940      ;
0950 001C 0011F0202100004F08000C004000 TST ALU FLAGS JSND FETCH
0960 001D 0011F0201700000C08006C203C00 ALU,REG,SRC,,ADDR,F,INS,CZERO FLAGS,ALL JNAP FETCH
0970      ;

0980      ;
0990      ; BIT TEST

```


NAVTRAEQUIPCEN 78-C-0157-1

PAGE 4 FIXED MASH VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV4.0

```

1000                                     ;      FORMAT: OPCODE MASK.RX, SRC.RX
1010                                     ;
1020 001E 0011F0202100004F08000C004000 BIT      ALU FLAGS JSND FETCH
1030 001F 0011F0202100004F08144C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND FETCH
1040 0020 0011F0202100000C0814CC001C00      ALU,REG,SRC,,DST,AND,F,NWR,CZERO FLAGS,UNZ JNAP FETCH
1050                                     ;

1060                                     ;
1070                                     ;      BIT SET
1080                                     ;      FORMAT: OPCODE MASK.RX,DST.MX (2 OPERANDS)
1090                                     ;      OPCODE MASK.RX, SRC.RX, DST.MX (3 OPERANDS)
1100                                     ;
1110 0021 0011F0202100004F08000C004000 BIS2      ALU FLAGS JSND FETCH
1120 0022 000000002100004F08144C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND
1130 0023 0091F0202300002F0814FC200C00      ALU,REG,SRC,,DST,OR,F,INS,CZERO FLAGS,NZ LDOR JSND FETCH
1140                                     ;
1150 0024 0011F0202100004F08000C004000 BIS3      ALU FLAGS JSND FETCH
1160 0025 0011F0202100004F08144C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND FETCH
1170 0026 0091F0202300002F0814FC200C00      ALU,REG,SRC,,DST,OR,F,INS,CZERO FLAGS,NZ LDOR JSND FETCH
1180                                     ;

1190                                     ;
1200                                     ;      BIT CLEAR
1210                                     ;      FORMAT: OPCODE MASK.RX, DST.MX (2 OPERANDS)
1220                                     ;      OPCODE MASK.RX, SRC.RX, DST.MX (3 OPERANDS)
1230                                     ;
1240 0027 0011F0202100004F08000C004000 BIC2      ALU FLAGS JSND FETCH
1250 0028 000000002100004F08147C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND
1260 0029 0091F0202300002F0814CC200C00      ALU,REG,SRC,,DST,AND,F,INS,CZERO FLAGS,NZ LDOR JSND FETCH
1270                                     ;
1280 002A 0011F0202100004F08000C004000 BIC3      ALU FLAGS JSND FETCH
1290 002B 0011F0202100004F08147C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND FETCH
1300 002C 0091F0202300002F0814CC200C00      ALU,REG,SRC,,DST,AND,F,INS,CZERO FLAGS,NZ LDOR JSND FETCH
1310                                     ;

1320                                     ;
1330                                     ;      EXCLUSIVE OR
1340                                     ;      FORMAT: OPCODE MASK.RX, DST.MX (2 OPERANDS)
1350                                     ;      OPCODE MASK.RX, SRC.RX, DST.MX (3 OPERANDS)
1360                                     ;
1370 002D 0011F0202100004FC000C004000 XOR2      ALU FLAGS JSND FETCH
1380 002E 000000002100004F08144C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND
1390 002F 0011F0202300002F08148C200C00      ALU,REG,SRC,,DST,EXOR,F,INS,CZERO FLAGS,NZ JSND FETCH
1400                                     ;
1410 0030 0011F0202100004F08000C004000 XOR3      ALU FLAGS JSND FETCH
1420 0031 0011F0202100004F08144C200000      ALU,REG,SRC,,DST,ADDR,F,INS,CZERO FLAGS JSND FETCH
1430 0032 0011F0202300002F08148C200C00      ALU,REG,SRC,,DST,EXOR,F,INS,CZERO FLAGS,NZ JSND FETCH
1440                                     ;

1450                                     ;
1460                                     ;      ARITHMETIC SHIFT
1470                                     ;      FORMAT: OPCODE CNT.RB, SRC.RL, DST.ML
1480                                     ;
1490 0033 000000000E00000008178C1E0000 ASH      ALU,...,LENGTH,ZERO,F,LWRD,CZERO FLAGS CONT

```

| PC | OP | OP1 | OP2 | OP3 | OP4 | OP5 | OP6 | OP7 | OP8 | OP9 | OP10 | OP11 | OP12 | OP13 | OP14 | OP15 | OP16 | OP17 | OP18 | OP19 | OP20 | OP21 | OP22 | OP23 | OP24 | OP25 | OP26 | OP27 | OP28 | OP29 | OP30 | OP31 | OP32 | OP33 | OP34 | OP35 | OP36 | OP37 | OP38 | OP39 | OP40 | OP41 | OP42 | OP43 | OP44 | OP45 | OP46 | OP47 | OP48 | OP49 | OP50 | OP51 | OP52 | OP53 | OP54 | OP55 | OP56 | OP57 | OP58 | OP59 | OP60 | OP61 | OP62 | OP63 | OP64 | OP65 | OP66 | OP67 | OP68 | OP69 | OP70 | OP71 | OP72 | OP73 | OP74 | OP75 | OP76 | OP77 | OP78 | OP79 | OP80 | OP81 | OP82 | OP83 | OP84 | OP85 | OP86 | OP87 | OP88 | OP89 | OP90 | OP91 | OP92 | OP93 | OP94 | OP95 | OP96 | OP97 | OP98 | OP99 | OP100 | OP101 | OP102 | OP103 | OP104 | OP105 | OP106 | OP107 | OP108 | OP109 | OP110 | OP111 | OP112 | OP113 | OP114 | OP115 | OP116 | OP117 | OP118 | OP119 | OP120 | OP121 | OP122 | OP123 | OP124 | OP125 | OP126 | OP127 | OP128 | OP129 | OP130 | OP131 | OP132 | OP133 | OP134 | OP135 | OP136 | OP137 | OP138 | OP139 | OP140 | OP141 | OP142 | OP143 | OP144 | OP145 | OP146 | OP147 | OP148 | OP149 | OP150 | OP151 | OP152 | OP153 | OP154 | OP155 | OP156 | OP157 | OP158 | OP159 | OP160 | OP161 | OP162 | OP163 | OP164 | OP165 | OP166 | OP167 | OP168 | OP169 | OP170 | OP171 | OP172 | OP173 | OP174 | OP175 | OP176 | OP177 | OP178 | OP179 | OP180 | OP181 | OP182 | OP183 | OP184 | OP185 | OP186 | OP187 | OP188 | OP189 | OP190 | OP191 | OP192 | OP193 | OP194 | OP195 | OP196 | OP197 | OP198 | OP199 | OP200 | OP201 | OP202 | OP203 | OP204 | OP205 | OP206 | OP207 | OP208 | OP209 | OP210 | OP211 | OP212 | OP213 | OP214 | OP215 | OP216 | OP217 | OP218 | OP219 | OP220 | OP221 | OP222 | OP223 | OP224 | OP225 | OP226 | OP227 | OP228 | OP229 | OP230 | OP231 | OP232 | OP233 | OP234 | OP235 | OP236 | OP237 | OP238 | OP239 | OP240 | OP241 | OP242 | OP243 | OP244 | OP245 | OP246 | OP247 | OP248 | OP249 | OP250 | OP251 | OP252 | OP253 | OP254 | OP255 | OP256 | OP257 | OP258 | OP259 | OP260 | OP261 | OP262 | OP263 | OP264 | OP265 | OP266 | OP267 | OP268 | OP269 | OP270 | OP271 | OP272 | OP273 | OP274 | OP275 | OP276 | OP277 | OP278 | OP279 | OP280 | OP281 | OP282 | OP283 | OP284 | OP285 | OP286 | OP287 | OP288 | OP289 | OP290 | OP291 | OP292 | OP293 | OP294 | OP295 | OP296 | OP297 | OP298 | OP299 | OP300 | OP301 | OP302 | OP303 | OP304 | OP305 | OP306 | OP307 | OP308 | OP309 | OP310 | OP311 | OP312 | OP313 | OP314 | OP315 | OP316 | OP317 | OP318 | OP319 | OP320 | OP321 | OP322 | OP323 | OP324 | OP325 | OP326 | OP327 | OP328 | OP329 | OP330 | OP331 | OP332 | OP333 | OP334 | OP335 | OP336 | OP337 | OP338 | OP339 | OP340 | OP341 | OP342 | OP343 | OP344 | OP345 | OP346 | OP347 | OP348 | OP349 | OP350 | OP351 | OP352 | OP353 | OP354 | OP355 | OP356 | OP357 | OP358 | OP359 | OP360 | OP361 | OP362 | OP363 | OP364 | OP365 | OP366 | OP367 | OP368 | OP369 | OP370 | OP371 | OP372 | OP373 | OP374 | OP375 | OP376 | OP377 | OP378 | OP379 | OP380 | OP381 | OP382 | OP383 | OP384 | OP385 | OP386 | OP387 | OP388 | OP389 | OP390 | OP391 | OP392 | OP393 | OP394 | OP395 | OP396 | OP397 | OP398 | OP399 | OP400 | OP401 | OP402 | OP403 | OP404 | OP405 | OP406 | OP407 | OP408 | OP409 | OP410 | OP411 | OP412 | OP413 | OP414 | OP415 | OP416 | OP417 | OP418 | OP419 | OP420 | OP421 | OP422 | OP423 | OP424 | OP425 | OP426 | OP427 | OP428 | OP429 | OP430 | OP431 | OP432 | OP433 | OP434 | OP435 | OP436 | OP437 | OP438 | OP439 | OP440 | OP441 | OP442 | OP443 | OP444 | OP445 | OP446 | OP447 | OP448 | OP449 | OP450 | OP451 | OP452 | OP453 | OP454 | OP455 | OP456 | OP457 | OP458 | OP459 | OP460 | OP461 | OP462 | OP463 | OP464 | OP4 |
|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----|
|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----|

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 1 FIXED HASH VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV6.0

```

2020
2030 0053 0011F0212100004F08000C004000 SUB1 ALU FLAGS JSND FETCH
2040 0054 0011F0201100004F08144C200000 ALU,REG, SRC, .DST, ADDR, F, INS, CZERO FLAG JSND FETCH
2050 0057 0091F0202300002F08142C203C00 ALU,REG, SRC, .DST, SUB, F, INS, CZERO FLAG, ALL LDOR JSND FETCH
2060
2070 0058 0011F0202100004F08000C004000 SBMC ALU FLAGS JSND FETCH
2080 0059 000000002100004F08144C1E0000 ALU,REG, SRC, .DST, ADDR, F, LMRD, CZERO FLAG JSND
2090 005A 0091F0202300002F08142C1EFC00 ALU,REG, SRC, .DST, SUB, F, LMRD, CYN FLAG, ALL LDOR JSND FETCH
2100

2110
2120
2130
2140
2150 005B 0011F0202100004F08000C004000 MULL2 ALU FLAGS JSND FETCH
2160 005C 000000002100004F08404C200000 ALU,REG, SRC, G, .ADDR, F, INS, CZERO FLAG JSND
2170 005D 000000004C001D000B140C200000 ALU,REG, ., .DST, ZERO, F, INS, CZERO FLAG LDCT, 001DN
2180 005E 000000004400000000B1402200004 SPALU,REG, SRC, DST, MULT2C, INS, CZERO, .SIO FLAG, RPCT
2190 005F 000000000E00000000B1404217804 SPALU,REG, SRC, DST, MULTC, INS, ZIN, .SIO FLAG, CYN CONT
2200 0060 0011F0202300002F08144C200400 ALU,REG, ., .DST, ADDS, F, INS, CZERO FLAG, Z JSND FETCH
2210

```


NAVTRAEQUIPCEN 78-C-0157-1

PAGE 8 FIXED MASH VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV6.0

```

2770
2780
2790 007C 00000000E0000000B178C1E0000 BRB
2800 007D 00000000E0000000B17FE3E0000
2810 007E 020000200E0000004B173C003C00
2820 007F 0015F0200E00000008000C004000
2830 0080 0011F0201700000C08000C004000
2840
2850
2860 0081 0011F0200E0000000B178C1E0000 BRW
2870 0082 0011F020000000004B17FC020000
2880 0083 00000000E00000004B17FEDC0000
2890 0084 020000200E0000004B173C1E3C00
2900 0085 0015F0200E00000008000C004000
2910 0086 0011F0201700000C08000C004000
2920
2930
2940 0087 0011F0200E0000000B178C1E0000 JMP
2950 0088 0011F0200E0000004B17FC020000
2960 0089 0011F0200E0000004B17FC040000
2970 008A 0011F0200E0000004B17FC080000
2980 008B 0011F0200E0000004B17FC100000
2990 008C 020000200E00000008000C004000
3000 008D 0015F0200E00000008000C004000
3010 008E 0011F0201700000C08000C004000
3020

3030
3040
3050
3060
3070
3080
3090 008F 00000000E0000000B178C1E0000 BSBB
3100 0090 0011F0200E0000004B17FE3E0000
3110 0091 00000000E00000004B1744C1E0000
3120 0092 010000000E0000000F0E1C1E0000
3130 0093 0003F0500E00000008000C004000
3140 0094 020000200E00000008000C004000
3150 0095 0015F0200E00000008000C004000
3160 0096 0011F0201700000C08000C004000
3170
3180
3190 0097 00000000E0000000B178C1E0000 BRW
3200 0098 0011F0200E0000004B17FC020000
3210 0099 0011F0200E0000004B17FEDC0000
3220 009A 00000000E00000004B1744C1E0000
3230 009B 010000000E0000000F0E1C1E0000
3240 009C 0003F0500E00000008000C004000
3250 009D 020000200E00000008000C004000
3260 009E 0015F0200E00000008000C004000
3270 009F 0011F0201700000C08000C004000
3280
3290
3300 00A0 00000000E0000000B178C1E0000 JSB
3310 00A1 0011F0200E0000004B17FC020000

```

BRANCH WITH BYTE-DISPLACEMENT
 ALU,...LENGTH,ZERO,F,LWRD,CZERO FLAGS CONT
 ALU,BSB,...LENGTH,OR,EXTBL,LWRD,CZERO FLAGS CONT
 ALU,ADB,...LENGTH,ADD,F,NWR,CZERO FLAGS,ALL LPC BPC CONT
 ALU FLAGS FORCEF CONT
 ALU FLAGS FETCH JMAP

BRANCH WITH WORD-DISPLACEMENT
 ALU,...LENGTH,ZERO,F,LWRD,CZERO FLAGS FETCH CONT
 ALU,BSB,...LENGTH,OR,F,B0,CZERO FLAGS FETCH
 ALU,BSB,...LENGTH,OR,EXTM,B321,CZERO FLAGS CONT
 ALU,ADB,...LENGTH,ADD,F,LWRD,CZERO FLAGS,ALL LPC BPC CONT
 ALU FLAGS FORCEF CONT
 ALU FLAGS FETCH JMAP

JUMP
 ALU,...LENGTH,ZERO,F,LWRD,CZERO FLAGS FETCH CONT
 ALU,BSB,...LENGTH,OR,F,B0,CZERO FLAGS FETCH CONT
 ALU,BSB,...LENGTH,OR,F,B1,CZERO FLAGS FETCH CONT
 ALU,BSB,...LENGTH,OR,F,B2,CZERO FLAGS FETCH CONT
 ALU,BSB,...LENGTH,OR,F,B3,CZERO FLAGS FETCH CONT
 ALU,...LENGTH,ADD,F,NWR,CZERO FLAGS LPC BPC CONT
 ALU FLAGS FORCEF CONT
 ALU FLAGS FETCH JMAP

BRANCH & JUMP TO SUBROUTINE
 FORMAT: OPCODE DSPL,RX (BRANCH)
 OPCODE DST,AS (JUMP)

BRANCH TO SUBROUTINE WITH BYTE-DISPLACEMENT
 ALU,...LENGTH,ZERO,F,LWRD,CZERO FLAGS CONT
 ALU,BSB,...LENGTH,OR,EXTBL,LWRD,CZERO FLAGS FETCH CONT
 ALU,ADB,...EA,ADDR,F,LWRD,CZERO FLAGS CONT
 ALU,REG,FOUR,...SP,SUBSR,F,LWRD,CZERO FLAGS LADR CONT
 ALU,REG,EA,...ADDR,F,NWR,CZERO FLAGS WRITE,LWORD CONT
 ALU,REG,LENGTH,...EA,ADD,F,LWRD,CZERO FLAGS,ALL LPC BPC CONT
 ALU FLAGS FORCEF CONT
 ALU FLAGS FETCH JMAP

BRANCH TO SUBROUTINE WITH WORD-DISPLACEMENT
 ALU,...LENGTH,ZERO,F,LWRD,CZERO FLAGS CONT
 ALU,BSB,...LENGTH,OR,F,B0,CZERO FLAGS FETCH CONT
 ALU,BSB,...LENGTH,OR,EXTM,B321,CZERO FLAGS FETCH CONT
 ALU,ADB,...EA,ADDR,F,LWRD,CZERO FLAGS CONT
 ALU,REG,FOUR,...SP,SUBSR,F,LWRD,CZERO FLAGS LADR CONT
 ALU,REG,EA,...ADDR,F,NWR,CZERO FLAGS WRITE,LWORD CONT
 ALU,REG,LENGTH,...EA,ADD,F,LWRD,CZERO FLAGS,ALL LPC BPC CONT
 ALU FLAGS FORCEF CONT
 ALU FLAGS FETCH JMAP

JUMP TO SUBROUTINE
 ALU,...LENGTH,ZERO,F,LWRD,CZERO FLAGS CONT
 ALU,BSB,...LENGTH,OR,F,B0,CZERO FLAGS FETCH CONT

400 JOURNAL OF POST KEYNESIAN ECONOMICS

[illegible]

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 10 FIXED MASH VAX EMULATOR MICROPROGRAM: INTEGER INSTRUCTIONS REV.0

| | | | | |
|------|------|------------------------------|--------|---|
| 3850 | 00C7 | 0011F0200E0000004816FC020000 | | ALU,BSB,...EA,OR,F,DO,CZERO FLAGS FETCH CONT |
| 3860 | 00C8 | 0011F0200E0000004816FEDE0000 | | ALU,BSB,...EA,OR,EXTM,LWRD,CZERO FLAGS FETCH CONT |
| 3870 | 00C9 | 020000200E0000004816JC000000 | | ALU,ADD,...EA,ADD,F,NWR,CZERO FLAGS LPC BPC CONT |
| 3880 | 00CA | 0013F0200E00000008000C004000 | | ALU FLAGS FORCEF CONT |
| 3890 | 00CB | 0011F0201700000C0A004C001C00 | | ALU,REG,DST,...ADDR,F,NWR,CZERO FLAGS,VNZ FETCH JMAP |
| 3900 | 00CC | 000000004300C0108000C004000 | ACBXA | ALU FLAGS CMP,ACBXB,GEQ |
| 3910 | 00CD | 0011F0200E00000008000C004000 | | ALU FLAGS FETCH CONT |
| 3920 | 00CE | 0011F0200E00000008000C004000 | | ALU FLAGS FETCH CONT |
| 3930 | 00CF | 0011F0201700000C0A004C001C00 | | ALU,REG,DST,...ADDR,F,NWR,CZERO FLAGS,VNZ FETCH JMAP |
| 3940 | | | | |
| 3950 | | | | |
| 3960 | | | | ADD ONE AND BRANCH |
| 3970 | | | | FORMAT: OPCODE LIMIT,RL,INDEX,HL,DISPL,BB |
| 3980 | | | | |
| 3990 | 00D0 | 0011F0202100004F08000C004000 | AOBLS | ALU FLAGS FETCH JSMD |
| 4000 | 00D1 | 000000002100004F08174C1E0000 | | ALU,REG,SRG,...LENGTH,ADDR,F,LWRD,CZERO FLAGS JSMD |
| 4010 | 00D2 | 080000000E00000008144C1E0000 | | ALU,REG,SRG,...DST,ADDR,F,LWRD,CZERO FLAGS SCNT CONT |
| 4020 | 00D3 | 0011F0202300002F08144C1E4000 | | ALU,REG,...DST,ADD,F,LWRD,CONE FLAGS FETCH JSMD |
| 4030 | 00D4 | 000000000E0000000814C001C00 | | ALU,REG,LENGTH,...DST,SUBSR,F,NWR,CZERO FLAGS,VNZ CONT |
| 4040 | 00D5 | 000000004300D901E8148C1E0000 | | ALU,NOSRC,...EA,ZERO,F,LWRD,CZERO FLAGS CJMP,AOBLSA,GEQ |
| 4050 | 00D6 | 020000200E0000004816FE3E0000 | | ALU,BSB,...EA,OR,EXTBL,LWRD,CZERO FLAGS LPC BPC CONT |
| 4060 | 00D7 | 0013F0200E00000008000C004000 | | ALU FLAGS FORCEF CONT |
| 4070 | 00D8 | 0011F0201700000C0A004C001C00 | | ALU,REG,DST,...ADDR,F,NWR,CZERO FLAGS,VNZ FETCH JMAP |
| 4080 | 00D9 | 0011F0200E00000008000C004000 | AOBLSA | ALU FLAGS FETCH CONT |
| 4090 | 00DA | 0011F0201700000C0A004C001C00 | | ALU,REG,DST,...ADDR,F,NWR,CZERO FLAGS,VNZ FETCH JMAP |
| 4100 | | | | |
| 4110 | 00DB | 0011F0202100004F08000C004000 | AOBLQ | ALU FLAGS FETCH JSMD |
| 4120 | 00DC | 000000002100004F08174C1E0000 | | ALU,REG,SRG,...LENGTH,ADDR,F,LWRD,CZERO FLAGS JSMD |
| 4130 | 00DD | 080000000E00000008144C1E0000 | | ALU,REG,SRG,...DST,ADDR,F,LWRD,CZERO FLAGS SCNT CONT |
| 4140 | 00DE | 0011F0202300002F08144C1E4000 | | ALU,REG,...DST,ADD,F,LWRD,CONE FLAGS FETCH JSMD |
| 4150 | 00DF | 000000000E00000008172C001C00 | | ALU,REG,DST,...LENGTH,SUB,F,NWR,CZERO FLAGS,VNZ CONT |
| 4160 | 00E0 | 000000004300E404E8148C1E0000 | | ALU,NOSRC,...EA,ZERO,F,LWRD,CZERO FLAGS CJMP,AOBLQA,OTR |
| 4170 | 00E1 | 020000200E0000004816FE3E0000 | | ALU,BSB,...EA,OR,EXTBL,LWRD,CZERO FLAGS LPC BPC CONT |
| 4180 | 00E2 | 0013F0200E00000008000C004000 | | ALU FLAGS FORCEF CONT |
| 4190 | 00E3 | 0011F0201700000C0A004C001C00 | | ALU,REG,DST,...ADDR,F,NWR,CZERO FLAGS,VNZ FETCH JMAP |
| 4200 | 00E4 | 0011F0200E00000008000C004000 | AOBLQA | ALU FLAGS FETCH CONT |
| 4210 | 00E5 | 0011F0201700000C0A004C001C00 | | ALU,REG,DST,...ADDR,F,NWR,CZERO FLAGS,VNZ FETCH JMAP |
| 4220 | | | | |
| 4230 | | | | |
| 4240 | | | | SUBTRACT ONE AND BRANCH |
| 4250 | | | | FORMAT: OPCODE INDEX,HL,DISPL,BB |
| 4260 | | | | |
| 4270 | 00E6 | 080000002100004F08000C004000 | S0BGE | ALU FLAGS SCNT JSMD |
| 4280 | 00E7 | 000000000E00000008144C1E0000 | | ALU,REG,SRG,...DST,ADDR,F,LWRD,CZERO FLAGS CONT |
| 4290 | 00E8 | 0011F0202300002F0814C1E3C00 | | ALU,NOSRC,...DST,SUBSR,F,LWRD,CONE FLAGS,VNZ FETCH JSMD |
| 4300 | 00E9 | 000000004300E321E8148C1E0000 | | ALU,NOSRC,...EA,ZERO,F,LWRD,CZERO FLAGS CJMP,S0BGEA,LSS |
| 4310 | 00EA | 020000200E0000004816FE3E0000 | | ALU,BSB,...EA,OR,EXTBL,LWRD,CZERO FLAGS LPC BPC CONT |
| 4320 | 00EB | 0013F0200E00000008000C004000 | | ALU FLAGS FORCEF CONT |
| 4330 | 00EC | 0011F0201700000C0A004C001C00 | | ALU FLAGS FETCH JMAP |
| 4340 | 00ED | 0011F0200E00000008000C004000 | S0BGEA | ALU FLAGS FETCH CONT |
| 4350 | 00EE | 0011F0201700000C08000C004000 | | ALU FLAGS FETCH JMAP |
| 4360 | | | | |
| 4370 | 00EF | 080000002100004F08000C004000 | S0BGT | ALU FLAGS SCNT JSMD |
| 4380 | 00F0 | 000000000E00000008144C1E0000 | | ALU,REG,SRG,...DST,ADDR,F,LWRD,CZERO FLAGS CONT |

NAME _____ PHONE _____

11. *Parasitica* (100%)

| | | |
|-----|------|----------------------------------|
| 100 | 00F1 | 00111001000000000000000000000000 |
| 100 | 00F2 | 000000000400F0000000000000000000 |
| 100 | 00F3 | 020000200F0000000000000000000000 |
| 100 | 00F4 | 00000000000000000000000000000000 |
| 100 | 00F5 | 00111001000000000000000000000000 |
| 100 | 00F6 | 00111001000000000000000000000000 |
| 100 | 00F7 | 00111001000000000000000000000000 |

[illegible]

3. 9. 3

4. 1. 4

4. 2. 4

```

      DO I=1,NP-1
        CALL DDISPL(I,XPOS(X),XNEG(X),DISPL(0I,BW,...,DISPL(LIM))

```

[illegible][illegible]

TAL FRANKS: 0

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 1 FLOATING VAX EMULATOR ASSEMBLY FILE.FLOATING POINT

```

0010
0020
0030
0040
0050
0060
0070 0000 000000000E00000008128C1E4000 INIT
0080 0001 000000000E00000008138C1E4000
0090
0100 0002 000004000E0000000810A9DE0000 /
0110
0120 0003 000008000E000000081469DE0000 /
0130 0004 000000000E0000000812AC040000
0140 0005 000000000E0000000813AC040000
0150 0006 000000000E00000008108C044000
0160 0007 000000000E00000008148C044000
0170 0008 000000000E0000000810FC1E4000
0180 0009 000000000E0000000814FC1E4000
0190 000A 000000000E000000081039DE0000
0200 000B 000000000E000000081439DE0000
0210
0220
0230
0240 000C 0400000004300103009132C007C00 EQUALE
0250 000D 0000000004300152108000C004000
0260 000E 000000000E000000081469DE0000
0270
0280 000F 0000000843000F10081441DE0000 /
0290 0010 000000000E0000000812AC040000 ADD2FC
0300 0011 000000000E000000081041DE0010 ADD2FA
0310 0012 000000000E000000081441DE0018 ADD2FE
0320 0013 000000000E000000081005DF0000
0330 0014 000000000E000000081405DF0000
0340 0015 000000000E000000081049DE0000 ADD2FB
0350 ADD2FD
0360 0016 00000000043001030081041DE0000
0370 0017 0000000004300112F099261DE0000
0380
0390
0400
0410 0018 0019F0202300003108000C004000 NORMAL
0420 0019 0019F020230000310814041E4C00
0430 001A 000000000E000000081405DF0000
0440 001B 000000000E0000000814481E0000
0450 001C 000000000E0000000814481E0000
0460 001D 000000000E0000000814481E0000
0470 001E 000000000E0000000814481E0000
0480 001F 00000000043002D320814481E0000
0490 0020 000000000430024320814481E0000
0500 0021 000000000E00000008121C1E4400 ADD2FI
0510 0022 0000000004300302008000C007C00
0520 0023 000000000430021320814481E0000
0530 0024 000000000E0000000814401E0000 ADD2FO
0540 0025 000000000E0000000814401E0000
0550 0026 000000000E0000000814401E0000
0560 0027 000000000E0000000814401E0000
0570 0028 000000000E0000000814401E0000
0580 0029 000000000E0000000814401E0000

TITLE VAX EMULATOR ASSEMBLY FILE.FLOATING POINT
OPT T=DFTA.DT:1:L=FLOATING.AL:1:N=120
ORG 00H

FLOATING POINT INITIALIZATION
ALU...SRC3,ZERO,F,LWRD,FLAGS CONT
ALU...SRC4,ZERO,F,LWRD,FLAGS CONT
ALU,REG,SRC,,SRC,ADDR,FLU,LWRD,CZERO,SHFTO,FLAGS,FSIGN,SIOM1
CONT
ALU,REG,DST,,DST,ADDR,FLU,LWRD,CZERO,SHFTO,FLAGS,FSIGN,SIOM1
CONT
ALU,REG,SRC,,SRC3,ADDR,F,B1,CZERO,FLAGS CONT
ALU,REG,DST,,SRC4,ADDR,F,B1,CZERO,FLAGS CONT
ALU...SRC,ZERO,F,B1,FLAGS CONT
ALU...DST,ZERO,F,B1,FLAGS CONT
ALU,CONST,,,SRC,OR,F,LWRD,FLAGS PIPE,0100H CONT
ALU,CONST,,,DST,OR,F,LWRD,FLAGS PIPE,0100H CONT
ALU,REG,SRC,,SRC,ADDR,FLU,LWRD,CZERO,FLAGS CONT
ALU,REG,DST,,DST,ADDR,FLU,LWRD,CZERO,FLAGS RET

FLOATING POINT EXPONENT EQUALIZATION
ALU,REG,SRC3,,SRC4,SUB,F,NWR,CONE,FLAGS,ALL,LEC,CJMP,ADD2FA,MORE
ALU,FLAGS,CJMP,ADD2FB,LSS
ALU,REG,DST,,DST,ADDR,FLU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,LWRD,CZERO,SHFTO,FLAGS,EXPCD
CJMP,ADD2FC,NMORED
ALU,REG,,,SRC3,ADD,F,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,SRC,ADD,F,LWRD,CZERO,SSFFO,FLAGS CONT
ALU,REG,,,DST,ADD,F,LWRD,CZERO,SSFFI,FLAGS CONT
SPALU,REG,,,SRC,CONV,LWRD,ZIN,FLAGS CONT
SPALU,REG,,,DST,CONV,LWRD,ZIN,FLAGS RET
ALU,REG,,,SRC,ADD,F,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,SRC,ADD,F,LWRD,CZERO,SHFTO,FLAGS,EXPCD
CJMP,ADD2FD,NMORED
ALU,REG,SRC4,,SRC3,ADDR,FLU,LWRD,CZERO,SHFTO,FLAGS,JUMP,ADD2FE

FLOATING POINT NORMALIZATION
ALU,FLAGS,CJMPD,ALUO,CFETCH
SPALU,REG,,,DST,INCR,LWRD,CONE,FLAGS,NZ,CJMPD,ALUO,CFETCH
SPALU,REG,,,DST,CONV,LWRD,ZIN,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS,CJMP,ADD2FJ,81013
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS,CJMP,ADD2FB,81013
ALU,CONST,,,SRC3,SUB,F,LWRD,CONE,FLAGS,Z,PIPE,0000H CONT
ALU,FLAGS,ALL,CJMP,UNDER,EQL
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS,CJMP,ADD2FI,81013
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT
ALU,REG,,,DST,ADD,F,FAU,LWRD,CZERO,SHFTO,FLAGS CONT

```

100

[illegible]

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 3 FLOATING MASH VAX EMULATOR ASSEMBLY FILE.FLOATING POINT

| | | | |
|------|------|------------------------------|--|
| 1170 | | | ALU.REG...SRC.ADDS.FLD.LWRD.CZERO.SSFFO FLAGS.ALL |
| 1180 | 0055 | 0019F02023000030081041DE3C10 | CJUMP.MOREO CFETCH |
| 1190 | 0054 | 000000000E000000081005DF0000 | SPALU.REG...SRC.CONV.LWRD.ZIN FLAGS CONT |
| 1200 | 0057 | 000000000E000000081405DF0000 | SPALU.REG...DST.CONV.LWRD.ZIN FLAGS CONT |
| 1210 | 0058 | 000000000E0000000A15AC200000 | ALU.REG.DST...DST2.ADDR.F.INS.CZERO FLAGS CONT |
| 1220 | 0059 | 000000000E0000000818AC200000 | ALU.REG.SRC...INDEX.ADDR.F.INS.CZERO FLAGS CONT |
| 1230 | 005A | 000000004300A9220818051F0000 | SPALU.REG...INDEX.CONV.LWRD.ZIN FLAGS CJMP.DIV2FA.VS |
| 1240 | 005B | 0000000043005F22081305DF0000 | SPALU.REG...DST2.CONV.LWRD.ZIN FLAGS CJMP.DIV2FE.VS |
| 1250 | 005C | 000000000E0000000A9549DE0000 | ALU.REG.DST2...DST2.ADDR.FLU.LWRD.CZERO FLAGS CONT |
| 1260 | 005D | 000000000E0000000C1849DE0000 | ALU.REG.INDEX...INDEX.ADDR.FLU.LWRD.CZERO FLAGS CONT |
| 1270 | | | ALU.REG.INDEX...DST2.SUBSR.F.LWRD.CONE FLAGS.ALL |
| 1280 | 005E | 0000000043006C210C151C1E7C00 | CJMP.DIV2FC.LSS |
| 1290 | 005F | 000000000E00000008408A204000 | ALU.REG...DST.ZERO.PFQ.INS FLAGS CONT |
| 1300 | 0060 | 000000000E00180008143C1E0000 | ALU.CONST...DST.ADD.F.LWRD.CZERO FLAGS PIPE.0018M CONT |
| 1310 | 0061 | 000000004400182F08000C004000 | ALU FLAGS PUSH.0018M.TRUE |
| 1320 | 0062 | 000000000800008008140C1F3C00 | SPALU.REG.SRC.DST.DIV2C.LWRD.ZIN FLAGS.ALL REPLOP |
| 1330 | 0063 | 000000000E00000008140E1F0000 | SPALU.REG.SRC.DST.D2CC.LWRD.ZIN FLAGS CONT |
| 1340 | 0064 | 000000000E00000008544C1E0000 | ALU.REG...DST.ADDS.F.LWRD.CZERO FLAGS CONT |
| 1350 | 0065 | 000000000E00000009921C1E4000 | ALU.REG.SRC4...SRC3.SUBSR.F.LWRD.CONE FLAGS CONT |
| 1360 | 0066 | 100000000E00000008123C1E0000 | ALU.CONST...SRC3.ADD.F.LWRD.CZERO FLAGS Z PIPE.0080M CONT |
| 1370 | 0067 | 000000004300322208000C005000 | ALU FLAGS V CJMP.OVER.VS |
| 1380 | 0068 | 000000004300282F08000C004000 | ALU FLAGS JUMP.NORMAL |
| 1390 | 0069 | 000000000E0000000810401E0000 | ALU.REG...SRC.ADDS.FAD.LWRD.CZERO.SHFTO FLAGS CONT |
| 1400 | 006A | 000000000E00000008121C1E4400 | ALU.CONST...SRC3.SUBSR.F.LWRD.CONE FLAGS Z PIPE.0080M CONT |
| 1410 | 006B | 000000004300592F08000C004000 | ALU FLAGS JUMP.DIV2FG |
| 1420 | 006C | 000000000E000000081449DE0000 | ALU.REG...DST.ADDS.FLU.LWRD.CZERO.SHFTO FLAGS CONT |
| 1430 | 006D | 000000000E00000008133C1E0400 | ALU.CONST...SRC4.ADD.F.LWRD.CZERO FLAGS Z PIPE.0080M CONT |
| 1440 | 006E | 000000000E0000000C152C1E4000 | ALU.REG.INDEX...DST2.SUB.F.LWRD.CONE FLAGS CONT |
| 1450 | 006F | 0000000043005E2F081549DE0000 | ALU.REG...DST2.ADDS.FLU.LWRD.CZERO.SHFTO FLAGS JUMP.DIV2FM |
| 1460 | 0070 | 000000000E00000008158C1E4000 | ALU...DST2.ZERO.F.LWRD FLAGS CONT |
| 1470 | 0071 | 000000000E00000008148C1E4000 | ALU...DST.ZERO.F.LWRD FLAGS CONT |
| 1480 | 0072 | 000000000E0000000814FC1E4000 | ALU.CONST...DST.OR.F.LWRD FLAGS PIPE.0000M CONT |
| 1490 | 0073 | 0011F0202300002F08000C004000 | ALU FLAGS JUMP.FETCH |
| 1500 | | | |
| 1510 | | | FLOATING POINT INITIALIZATION(DOUBLE PRECISION) |
| 1520 | | | |
| 1530 | 0074 | 000000000E00000008128C1E4000 | INITD ALU...SRC3.ZERO.F.LWRD FLAGS CONT |
| 1540 | 0075 | 000000000E00000008554C1E0000 | ALU.REG...DST2.ADDS.F.LWRD.CZERO FLAGS CONT |
| 1550 | 0076 | 000000000E00000008138C1E4000 | ALU...SRC4.ZERO.F.LWRD FLAGS CONT |
| 1560 | 0077 | 000000000E000000081146200000 | ALU.REG...SRC2.ADDS.PFQ.INS.CZERO FLAGS CONT |
| 1570 | | | ALU.REG...SRC.ADDS.FLUQ.LWRD.CZERO.SHFTO.SIO FLAGS |
| 1580 | 0078 | 000004000E000000081048DE0004 | FSIGN.SIGNO CONT |
| 1590 | 0079 | 000000000E00000008514C1E0000 | ALU.REG...DST.SRC2.ADDS.F.LWRD.CZERO FLAGS CONT |
| 1600 | 007A | 000000000E000000081346200000 | ALU.REG...DST2.ADDS.PFQ.INS.CZERO FLAGS CONT |
| 1610 | | | ALU.REG...DST.ADDS.FLUQ.LWRD.CZERO.SHFTO.SIO FLAGS |
| 1620 | 007B | 000000000E000000081448DE0004 | FSIGN.SIGN1 CONT |
| 1630 | 007C | 000000000E0000000812AC040000 | ALU.REG.SRC...SRC3.ADDR.F.B1.CZERO FLAGS CONT |
| 1640 | 007D | 000000000E0000000A13AC040000 | ALU.REG.DST...SRC4.ADDR.F.B1.CZERO FLAGS CONT |
| 1650 | 007E | 000000000E00000008108C044000 | ALU...SRC.ZERO.F.B1 FLAGS CONT |
| 1660 | 007F | 000000000E00000008148C044000 | ALU...DST.ZERO.F.B1 FLAGS CONT |
| 1670 | 0080 | 000000000E0100000810FC1E4000 | ALU.CONST...SRC.OR.F.LWRD FLAGS PIPE.0100M CONT |
| 1680 | 0081 | 000000000E0100000814FC1E4000 | ALU.CONST...DST.OR.F.LWRD FLAGS PIPE.0100M CONT |
| 1690 | 0082 | 000000000E000000081448DE0004 | ALU.REG...DST.ADDS.FLUQ.LWRD.CZERO.SHFTO.SIO FLAGS CONT |
| 1700 | 0083 | 000000000E000000081448DE0004 | ALU.REG...DST2.ADDS.FLUQ.LWRD.CZERO.SHFTO.SIO FLAGS CONT |
| 1710 | 0084 | 000000000E00000008554C1E0000 | ALU.REG...DST2.ADDS.F.LWRD.CZERO FLAGS CONT |
| 1720 | 0085 | 000000000E000000081146200000 | ALU.REG...SRC2.ADDS.PFQ.INS.CZERO FLAGS CONT |
| 1730 | 0086 | 000000000E000000081048DE0004 | ALU.REG...SRC.ADDS.FLUQ.LWRD.CZERO.SHFTO.SIO FLAGS CONT |
| 1740 | 0087 | 000000000E000000081048DE0004 | ALU.REG...SRC.ADDS.FLUQ.LWRD.CZERO.SHFTO.SIO FLAGS CONT |

[illegible]

NAVTRAEQUIPCEN 78-C-0157-1

2320 00B4 000000000E0000000B14421E0004

ALU.REG...DST,ADDS,FADQ,LWRD,CZERO,SHFT0,SIO FLAGS CONT

PAGE 5 FLOATING MASH VAX EMULATOR ASSEMBLY FILE.FLOATING POINT

2330 00B5 000000000E0000000B14421E0004

ALU.REG...DST,ADDS,FADQ,LWRD,CZERO,SHFT0,SIO FLAGS CONT

2340 00B6 000000000E0000000B534C1E0000

ALU.REG...Q,DST2,ADDS,F.LWRD,CZERO FLAGS CONT

2350 00B7 0011F0202300002F0914FC1E0000

ALU.REG.SRC3...DST,OR,F.LWRD,CZERO FLAGS JMD FETCH

2360 00B8 000000000E0000000B123C1E0000

ADD2F0

ALU.CONST...SRC3,ADD,F.LWRD,CZERO FLAGS PIPE,0000M CONT

2370 00B9 00000000430032210B000C004000

ALU FLAGS CUMP,OVER,LSB

2380 00BA 000000004300AE2F0B000C004000

ALU FLAGS JUMP,ADD2F0

2390

2400

2410

FLOATING POINT ADDITION (DOUBLE PRECISION)

2420 00BB 0011F0202100006F0B000C004000

ADD2FT

ALU FLAGS JSMD FETCH

2430 00BC 0011F0202100006F0B116C200000

ALU.REG.SRC...SRC2,ADDR,F.INS,CZERO FLAGS JSMD FETCH

2440 00BD 000000002100006F0A156C200000

ALU.REG.DST...DST2,ADDR,F.INS,CZERO FLAGS JSMD

2450 00BE 000000004100742F0B000C004000

ALU FLAGS JSB,INITD

2460 00BF 000000004100892F0B000C004000

ALU FLAGS JSB,EQUAED

2470

ALU.REG.SRC2...DST2,ADD,F.LWRD,CZERO FLAGS,ALL,AF,ZER,NZAF

2480 00C0 000000000E0000000B933C1E3C80

/CONT

ALU.REG.SRC...DST,ADD,F.LWRD,CY FLAGS,ALL,ZER,ZER,NZAF CONT

2490 00C1 000000000E0000000B143C1E9E80

ALU FLAGS JUMP,NORMLD

2500 00C2 0000000043009F2F0B000C004000

2510

FLOATING POINT SUBTRACTION (DOUBLE PRECISION)

2520

2530

2540 00C3 0011F0202100006F0B000C004000

SUB2FD

ALU FLAGS JSMD FETCH

2550 00C4 0011F0202100006F0B116C200000

ALU.REG.SRC...SRC2,ADDR,F.INS,CZERO FLAGS JSMD FETCH

2560 00C5 000000002100006F0A156C200000

ALU.REG.DST...DST2,ADDR,F.INS,CZERO FLAGS JSMD

2570 00C6 000000004100742F0B000C004000

ALU FLAGS JSB,INITD

2580 00C7 000000004100892F0B000C004000

ALU FLAGS JSB,EQUAED

2590 00C8 000000000E0000000B933C1E7F80

ALU.REG.SRC2...DST2,SUB,F.LWRD,CONE FLAGS,ALL,BOR,ZER,NZAF CONT

2600 00C9 000000000E0000000B142C1EFEB0

ALU.REG.SRC...DST,SUB,F.LWRD,CYN FLAGS,ALL,ZER,ZER,NZAF CONT

2610 00CA 0000000043009F2F0B000C004000

ALU FLAGS JUMP,NORMLD

2620

2630

2640

FLOATING POINT MULTIPLICATION (DOUBLE PRECISION)

2650 00CB 0011F0202100006F0B000C004000

MUL2FD

ALU FLAGS JSMD FETCH

2660 00CC 0011F0202100006F0B116C200000

ALU.REG.SRC...SRC2,ADDR,F.INS,CZERO FLAGS JSMD FETCH

2670 00CD 000000002100006F0A156C200000

ALU.REG.DST...DST2,ADDR,F.INS,CZERO FLAGS JSMD

2680 00CE 000000004100742F0B000C004000

ALU FLAGS JSB,INITD

2690 00CF 000000000E0000000B1546200000

ALU.REG...DST2,ADDS,PFQ,INS,CZERO FLAGS CONT

2700

ALU.REG...DST,ADDS,FLDQ,LWRD,CZERO,SSFF1,SIO FLAGS,ALL

2710 00D0 0019F020230000300B1443DE3C1C

/

CJMD,MOREO CFETCH

2720 00D1 000000000E0000000B534C1E0000

MUL2F0

ALU.REG...Q,DST2,ADDS,F.LWRD,CZERO FLAGS CONT

2730 00D2 000000000E0000000B1146200000

ALU.REG...SRC2,ADDS,PFQ,INS,CZERO FLAGS CONT

2740

ALU.REG...SRC,ADDS,FLDQ,LWRD,CZERO,SSFF0,SIO FLAGS,ALL

2750 00D3 0019F020230000300B1043DE3C14

/

CJMD,MOREO CFETCH

2760 00D4 000000000E0000000B534C1E0000

ALU.REG...Q,SRC2,ADDS,F.LWRD,CZERO FLAGS CONT

2770 00D5 000000000E0000000B1105DF0800

SPALU.REG...SRC2,CONV,LWRD,ZIN FLAGS,CY CONT

2780 00D6 000000000E0000000B1053C44000

ALU.REG...SRC,ADDS,LWRD,CY FLAGS CONT

2790 00D7 000000000E0000000B1305DF0800

SPALU.REG...DST2,CONV,LWRD,ZIN FLAGS,CY CONT

2800 00D8 000000000E0000000B1453C44000

ALU.REG...DST,ADDS,LWRD,CY FLAGS CONT

2810 00D9 000000000E0000000B186C1E0000

ALU.REG.DST...INDEX,ADDR,F.LWRD,CZERO FLAGS CONT

2820 00DA 000000000E0000000B99AC1E0000

ALU.REG.DST2...INDEX2,ADDR,F.LWRD,CZERO FLAGS CONT

2830 00DB 000000000E0000000B186C1E4000

ALU...INDEX3,ZERO,F.LWRD FLAGS CONT

2840 00DC 000000000E0000000B186C1E4000

ALU...INDEX4,ZERO,F.LWRD FLAGS CONT

2850 00DD 000000004400382F0B000C004000

ALU FLAGS PUSH,0038M,TRUE

2860 00DE 000000004300E10E0B154C1E0000

ALU.REG...DST2,ADDS,F.LWRD,CZERO FLAGS CUMP,SHFTN,NSITO

2870 00DF 000000000E0000000B99AC1E0000

ALU.REG.SRC2...INDEX4,ADD,F.LWRD,CZERO FLAGS,CY CONT

2880 00E0 000000000E0000000B143C1E8000

ALU.REG.DST...INDEX3,ADD,F.LWRD,CY FLAGS CONT

2890 00E1 000000000E0000000B1846200000

SHFTN

ALU.REG...INDEX4,ADDS,PFQ,INS,CZERO FLAGS CONT

2900 00E2 000000000E0000000B1A421E0004

ALU.REG...INDEX3,ADDS,FADQ,LWRD,CZERO,SHFT0,SIO FLAGS CONT

• • •

155

NAVTRAEQUIPCEN 78-C-0157-1

PAGE 7 FLOATING MASH VAX EMULATOR ASSEMBLY FILE FLOATING POINT

```

3499 0113 000000000E0000000148DE0004
3500 0114 000000000E000000014C1E3C00
3510 0115 000000000E0000000E934C1E0000
3520 0116 000000000E0000000F14C1E0000
3530 0117 000000000E00000009721C1E4000
3540 0118 100000000E00000008123C1E0000
3550 0119 00000000430032208000C0000000
3560 011A 0000000043009FF2F08000C004000
3570 011B 000000000E00000008146200000 DIV2FN
3580 011C 000000000E000000081021E3004
3590 011D 000000000E0000000814C1E4400
3600 011E 000000004300FBF08514C1E0000
3610 011F 000000000E000000081346200000 DIV2FT
3620 0120 000000000E00000008148DE0004
3630 0121 000000000E00000008934C1E0000
3640 0122 000000000E00000008133C1E0400
3650
3660 0123 000000000E00000009A2C1E7F30
3670 0124 000000000E000000012C1EFC00
3680 0125 000000000E00000001A4620000
3690 0126 000000000E00000008148DE0004
3700 0127 000000004301062F085A4C1E0000
3710
3720 0128 000000000E00000008931C1E7F80
3730 0129 000000000E0000000131C1EFC00
3740 012A 000000000E00000001D4620000
3750 012B 000000000E00000008148DE0004
3760 012C 000000004301122F085D4C1E0000
3770
ALU.REG.,,SRC,ADDS,FLUQ,LWRD,CZERO,SMFTO,SIO FLAGS CONT
ALU.REG.,Q,SRC2,ADDS,F,LWRD,CZERO FLAGS,ALL REPOP
ALU.REG,EIGHT.,,DST,ADDR,F,LWRD,CZERO FLAGS CONT
ALU.REG,FOUR.,,DST,ADDR,F,LWRD,CZERO FLAGS CONT
ALU.REG,SRC4.,,SRC3,SUBSR,F,LWRD,CONE FLAGS CONT
ALU.CONST.,,SRC3,ADD,F,LWRD,CZERO FLAGS Z PIPE,OOBOM CONT
ALU FLAGS,V CMP,OVER,VS
ALU FLAGS JUMP,NORMLD
ALU.REG.,,SRC2,ADDS,PFQ,INS,CZERO FLAGS CONT
ALU.REG.,,SRC,ADDS,FADD,LWRD,CZERO,SMFTO,SIO FLAGS CONT
ALU.CONST.,,SRC3,SUBSR,F,LWRD,CONE FLAGS,Z PIPE,OOBOM CONT
ALU.REG.,Q,SRC2,ADDS,F,LWRD,CZERO FLAGS JUMP,DIV2FN
ALU.REG.,,DST2,ADDS,PFQ,INS,CZERO FLAGS CONT
ALU.REG.,,DST,ADDS,FLUQ,LWRD,CZERO,SMFTO,SIO FLAGS CONT
ALU.REG.,Q,DST2,ADDS,F,LWRD,CZERO FLAGS CONT
ALU.CONST.,,SRC4,ADD,F,LWRD,CZERO FLAGS,Z PIPE,OOBOM CONT
ALU.REG,INDEX2.,,INDEX3,SUB,F,LWRD,CONE
FLAGS,ALL,BOR,ZER,NZAF CONT
ALU.REG,INDEX.,,INDEX4,SUB,F,LWRD,CYN FLAGS,ALL CONT
ALU.REG.,,INDEX3,ADDS,PFQ,INS,CZERO FLAGS CONT
ALU.REG.,,INDEX,ADDS,FLUQ,LWRD,CZERO,SMFTO,SIO FLAGS CONT
ALU.REG.,Q,INDEX3,ADDS,F,LWRD,CZERO FLAGS JUMP,DIV2FS
ALU.REG,SRC2.,,DST2,SUBSR,F,LWRD,CONE
FLAGS,ALL,BOR,ZER,NZAF CONT
ALU.REG,SRC.,,DST,SUBSR,F,LWRD,CYN FLAGS,ALL CONT
ALU.REG.,,EIGHT,ADDS,PFQ,INS,CZERO FLAGS CONT
ALU.REG.,,FOUR,ADDS,FLUQ,LWRD,CZERO,SMFTO,SIO FLAGS CONT
ALU.REG.,Q,EIGHT,ADDS,F,LWRD,CZERO FLAGS JUMP,DIV2FU
END

```

TOTAL ERRORS 0

APPENDIX E

The instructions of Appendix A of ATIL Document N74-125 are listed in Table E-1 of this appendix along with assembly notation for the corresponding VAX-11/780 machine instructions and addressing modes. Addressing mode information is given in Table E-2 for all addressing modes. Each table gives the number of microcycle steps required for a given operation. To obtain execution times, multiply by the microcycle time of 100×10^{-9} sec.

The following items should be noted:

1. For most instructions, the best case and worst case execution time is based on addressing mode differences.
2. All floating-point operations are data dependent and best and worst case execution times are based on this as well as addressing modes for operands.
3. The branch and index instruction execution time depends on whether the test condition is true or false. This is due to the addition of an offset to the program counter when a branch occurs.
4. The execution times for source and destination operands are different because the destination mode subroutines allow for the possibility of a conditional return to the main microprogram. This capability is required for instructions in which the destination mode subroutine is not the end of the main microprogram as in the add-compare-and-branch instruction.
5. Addressing mode execution times depend on a number of other factors including the size of the operand, whether or not a memory access is aligned on a long word boundary and on the use of indexed addressing.
6. The worst case of addressing mode execution time is for non-aligned, indexed, long word displacement deferred which requires 17 microcycles for a 4 byte source and 14 microcycles for a 4 byte destination. This case is denoted by "index" in Table E-1 for the operand specifier. It is assumed that this mode is used to specify each operand of that instruction.
7. For some instructions, execution times depend on two source operands (S1 and S2) and one destination operand, even for a two operand instruction. This is because the main microprogram obtains the two source operands by source mode subroutines and stores the result based on a call to a destination mode subroutine.

NAVTRAEQUIPCEN 78-C-0157-1

8. In Table E-1, "I" refers to the number of microcycles for the execution of the main microprogram, "S" refers to the number of microcycles for the source mode subroutine and "D" for the number in the destination mode subroutine.

Table E-1. Details of AIET Calculation

(Numbers represent microcycles)

| | Best Case | Worst Case |
|----------------------------|-------------------------------|-------------------------|
| Load | | |
| MOVL (Rn), Rm | I 1 S 3 D $\frac{5}{8}$ | 2 4 $\frac{3}{9}$ |
| Load-Double | I 3 | 3 |
| MOVQ (Rn), Rm | S 4 D 4 11 | 7 $\frac{4}{14}$ |
| Store | | |
| MOVL Rn, (Rm) | I 2 S 1 D $\frac{5}{8}$ | 2 1 $\frac{6}{9}$ |
| Store-Double | I 3 | 3 |
| MOVQ Rn, (Rm) | S 2 D 6 11 | 2 $\frac{8}{13}$ |
| Add/Subtract | I 3 | 3 |
| ADDL2 Rn, Rn (best case) | S1 1 | 17 |
| ADDL2 Indexed (worst case) | S2 1 D $\frac{3}{9}$ | 17 $\frac{19}{56}$ |

NAVTRAEQUIPCEN 78-C-0157-1

| | | |
|-----------------------|------------------|------------------|
| Add/Subtract-Floating | I 24 | 135 |
| ADDF2 Rn, Rm (best) | S1 1 | 17 |
| ADDF2 Indexed (worst) | S2 1 | 17 |
| | D $\frac{3}{29}$ | $\frac{19}{188}$ |
| Multiply | | |
| MULL2 Rn, Rm (best) | I 35 | 35 |
| MULL2 Indexed (worst) | S1 1 | 17 |
| | S2 1 | 17 |
| | D $\frac{3}{40}$ | $\frac{19}{88}$ |
| Multiply-Floating | I 50 | 136 |
| MULF2 Rn, Rm (best) | S1 1 | 17 |
| MULF2 Indexed (worst) | S2 1 | 17 |
| | D $\frac{3}{55}$ | $\frac{19}{189}$ |
| Divide | I 43 | 43 |
| DIVL2 Rn, Rm (best) | S1 1 | 17 |
| DIVL2 Indexed (worst) | S2 1 | 17 |
| | D $\frac{3}{48}$ | $\frac{19}{96}$ |
| Divide-Floating | I 57 | 327 |
| DIVF2 Rn, Rm (best) | S1 1 | 17 |
| DIVF2 Indexed (worst) | D $\frac{3}{62}$ | $\frac{19}{380}$ |

NAVTRAEQUIPCEN 78 C-0157-1

| | | |
|--------------------------|-------------------|-----------------|
| Logical | I 3 | 3 |
| XORL2 Rn, Rm | S1 1 | 17 |
| XORL2 Indexed | S2 1 | 17 |
| | D $\frac{3}{8}$ | $\frac{19}{56}$ |
| Shift - 5 places | I 16 | 16 |
| ASHL #5, Rn, Rn (best) | S1 2 | 2 |
| ASHL 35, Indexed (worst) | S2 1 | 17 |
| | D $\frac{3}{22}$ | $\frac{19}{54}$ |
| Compare | | |
| CMPL Rn, Rm (best) | I 3 | 3 |
| CMPL Indexed | S 1 | 17 |
| | D $\frac{3}{7}$ | $\frac{19}{39}$ |
| Branch | I 2 | 5 |
| BEQ | | |
| Index | I 11 | 13 |
| ACBL Rj, Rk, Rl, Rm | S1 1 | 17 |
| ACBL Indexed | S2 1 | 17 |
| | D 3 | 19 |
| | S3 $\frac{1}{17}$ | $\frac{17}{83}$ |

NAVTRAEQUIPCEN 78-C-0157-1

| | | |
|-------------------------|-----------------|-----------------|
| Register to Register | I 1 | 1 |
| MOVL Rn, Rm | S 1 | 1 |
| | D $\frac{3}{5}$ | $\frac{3}{5}$ |
| Miscellaneous | I 2 | 3 |
| CVTBL Rn, Rm (best) | S 1 | 17 |
| CVTBL Indexed (worst) | D $\frac{3}{6}$ | $\frac{19}{38}$ |
| Input/Output* | I 231 | 251 |
| Set up and 10 transfers | | |

*Based on following program

```

MOVL # address1, R1      ; device address
MOVL # address2, R2      ; memory address
MOVB # 10, R3            ; load counter
LOOP: MOVL (R2) +,(R1)    ; transfer
DEC R3                   ; decrement counter
BGTR LOOP                ; loop until finished

```

I = instruction microcycles

S, S1, S2, S3 = Source operand microcycles

D = destination operand microcycles

TABLE 2-2. SELECTED ADDRESSING MODES EXECUTION STEPS

| Mode | NON-INDEXED | | | |
|---------------|-----------------------|-------------------|---------------------------|-----------------------|
| | Aligned 1,2,4 byte | Aligned 8 byte | Not aligned 1,2,4 byte | Not aligned 8 byte |
| Rn 5 Source | 1 | 2 | 1 | 2 |
| 5 Destination | 3 | 4 | 3 | 4 |
| (Rn) 6 source | 3 | 4 | 4 | 7 |
| 6 Destination | 5 | 6 | 6 | 8 |
| (Rn)+8 Source | 4 | 5 | 5 | 8 |
| 8 Destination | 6 | 7 | 7 | 9 |
| <hr/> | | | | |
| | INDEXED | | | |
| | Aligned 4 byte | Aligned 8 byte | Not Aligned 4 byte | Not Aligned 8 byte |
| F Source | 15 | 17 | 17 | 21 |
| F Destination | 17 | 19 | 19 | 22 |
| <hr/> | | | | |

Program Counter

Immediate 2 for Byte, 3 for Word, 5 for Longword

APPENDIX F

Application Task Manager
PASCAL Version

0071.2 DARE CIST FEB 79

PROGRAM ATM :

```

(*****
(*****
(*****
(*****      A.T.M. PROGRAM, WRITTEN FOR THE H8309 MICRO
(*****      PROCESSOR. BECAUSE PASCAL DOES NOT HAVE SOME
(*****      OF THE FEATURES OF THE H8309, THIS PROGRAM
(*****      MAY NOT HANDLE STRUCTURES THE SAME AS THE
(*****      FINAL VERSION WRITTEN ON THE H8309, BY S.S.
(*****
(*****
(*****
(*****

```

CONST

```

EVMAX : 304      (* MAXIMUM ENTRIES IN EVENT QUEUE *)
SVMAX : 304      (* MAXIMUM NUMBER OF SHARED VARIABLES *)

```

TYPE

```

A-PTYPE:
B-PTYPE:
C-PTYPE:
E-PTYPE:
F-PTYPE:
G-PTYPE:
H-PTYPE:
AA-RECORD      (*****
    BYTE : INTEGER; (* THIS STRUCTURE SIMULATES MEMORY *)
    POINTER:H;      (* LOCATIONS THAT WOULD CONTAIN OBJECT *)
    NEXT : H;       (* CODE OF SAME TYPE. *)
END;              (*****
BB-RECORD:
CC-RECORD      (* JOB ID THE OBJECT CODE FOR A PROGRAM *)
    JOB : INTEGER;
END;
DD-RECORD:
EE-RECORD      (*****
    ID : INTEGER;   (* THIS IS THE ACTIVE TASK STRUCTURE. A *)
    ICBP : I;       (* POINTER TO CURRENT TASK IS STORED HERE*)
END;              (*****
GG-RECORD      (*****
    ID:              (* SHARED VARIABLE STRUCTURE. *)
    LENGTH : INTEGER; (* THE READY FLAG IS USED ONLY FOR *)
    LOCADDR:         (* ADDRESS. *)
    SMAADDR : H;     (*****
    RWFLAG:
    READY : BOOLEAN;
END;

```


NAVTRAEQUIPCEN 78-C-0157-1

```

EVENTQ=RECORD      (*****
    TIME,          (* THIS IS THE EVENT QUEUE STRUC- *)
    OPCODE,        (* TRUE, THE OPCODE TELLS THE *)
    ID      : INTEGER; (* OPERATION TO BE PERFORMED, THE *)
    NEXT    : DD;      (* TIME IS GIVEN IN ABSOLUTE TIME *)
END;               (*****

```

```

RETURN=RECORD      (*****

    HIGHBYTE,      (* ANOTHER REPRESENTATIVE OF *)
    LOWBYTE : INTEGER; (* MEMORY. *)
END;               (*****

```

```

FF=RECORD          (*****
    ESOS,          (* ENTIRE STATE ON STACK *)
    FIM,          (* FAST INTERRUPT MASK      000000000000 *)
    HC,           (* HALF CARRY          0 CONDITION *)
    IIM,          (* IRQ INTERRUPT MASK  0 CODES FOR *)
    NEGAT,        (* NEGATIVE            0 H/BO9 *)
    ZERO,         (* ZERO                000000000000 *)
    OVER,         (* OVERFLOW *)
    CB:BOOLEAN; (* CARRY BORROW *)
END;              (*****

```

```

STACK=RECORD       (*****
    CC : FF;        (* *)
    REGA,          (* INFORMATION ON THE SYSTEMS *)
    REGB,          (* STACK AFTER AN INTERRUPT. *)
    DP : INTEGER;  (* FIRST THING PUSHED IS THE PC *)
    X  : G;        (* FIRST THING PULLED IS THE CC *)
    Y  : G;        (* *)
    USP : F;       (*****
    PC  : H;
END;

```

```

TSYSQ=RECORD       (*****
    RUN,           (* *)
    ACTIVE: BOOLEAN; (* SYSTEM JOBS QUEUE. *)
    ID      : INTEGER; (* ID *)
    TCBP    : E;    (* 1 JOB *)
    NEXT    : A;    (* 2 FRAME ROUTINE *)
END;          (* 3 PROTECTION *)
              (* 4 TIMER (JOB) *)
              (* 5 ADD A JOB *)
              (* 6 PURGE A JOB *)
              (* *)
              (*****

```

NAVTRAEQUIPCEN 78-C-0157 1

```
TJOBQ-RECORD      (*****
ID,               (* JOB QUEUE STRUCTURE, USED FOR      *)
PRIO : INTEGER;  (* FAST LOOK UP BY JOB ID.             *)
PRIORP : C;      (*****
ICBP : E;
NEXT : R;
END;
```

```
IPRIO-RECORD      (*****
ICBP : E;        (* PRIORITY QUEUE STRUCTURE, CONTAINS *)
RKF : C;        (* ALL JOBS ORDER BY PRIORITY FROM *)
EDP : C;        (* HIGHEST TO LOWEST, RKF POINTS TO THE *)
PRIOR:INTEGER;  (* NEXT HIGHER PRIO. TASK, EDP POINTS *)
END;            (* TO THE NEXT LOWER PRIO TASK.      *)
                (*****
```

```
TICB-RECORD      (*****
ID,               (* TASK CONTROL BLOCK, ALL JOBS      *)
PRIO : INTEGER;  (* HAVE AN ENTRY.                     *)
RUN:BOOLEAN;     (*****
```

```
INTER:ACTIVE;
SYSI : BOOLEAN;
TOTIME;
TIMEREM : INTEGER;
STADDR;
ENDADDR : RR;
ORGS : E;
STACKA : E;
EQ : C;
END;
```

VAR

```
INX : R;
NSP,SE,USERSEVATASE : E;
STARTED,ENDED,TEMPD : C;
CURD,TEMPD : C;
STARTDD,ENDDD,TEMPDD : R;
STARTSYD,ENDSYD,TEMPSYD : A;
FLAG,EVENT,SYN,SCH : BOOLEAN;
FIRSTD,TEMPE,TEMPER : DD;
FREE : ARRAY [1..SUMAX] OF DD;
CUR,ARSTIME,INTER : INTEGER;
TIMERD,TIMERE : INTEGER;
ACTJOB : E;
FLAGOS,OSINTER,OSAVED : BOOLEAN;
CONTRG,EVENPTY,RESET : BOOLEAN;
USERFLAG : ARRAY [1..SUMAX] OF BOOLEAN;
SHVAR : ARRAY [1..SUMAX] OF GG;
```

```

(*****
(*)
(*)      SUBROUTINES FOR THE MAIN ROUTINE      (*)
(*)
(*****
(*)
(*)      RETURN FROM INTERRUPT                  (*)
(*****

```

```

PROCEDURE RTI;
BEGIN
END;

```

```

(*****
(*)
(*)      1. OSENTRY - ENTER OPERATING SYSTEM MODE      (*)
(*****

```

```

PROCEDURE OSENTRY;
BEGIN
  IF FLAGOS THEN
    OSINTER:=TRUE
  ELSE
    BEGIN
      FLAGOS:=TRUE;
      USERSP:=SP;
      SP:=ATMSP;
    END;
  END;

```

00/1.2 AAEC (1ST FEB 78)

```

END;

```

```

(*****
(*)
(*)      2. OSEXIT - LEAVE OPERATING SYSTEM            (*)
(*****

```

```

PROCEDURE OSEXIT;
BEGIN
  IF OSINTER AND NOT(SP@.REGA=SSP@.REGA) THEN
    RTI
  ELSE
    BEGIN
      OSINTER:=FALSE;
      FLAGOS:=FALSE;
      ATMSP:=SP;
      SP:=USERSP;
      CONTREG:=TRUE; (* CONTROL REG FOR PROTECTION *)
      RTI;
    END
  END;
END;

```

```
(*****
(*)
(*) 3. HALT (*)
(*****)
```

PROCEDURE AHALT;

```
  BEGIN
    REPEAT
      UNTIL 1 = 2 ;
  END;
```

```
(*****
(*)
(*) 4. COMMUNICATION ROUTINES (*)
(*) THE INITIALIZATION ROUTINE IS INCLUDED. (*)
(*****)
```

PROCEDURE INIT;

```
  BEGIN
    RESET := FALSE;
  END;
```

PROCEDURE COMM;

```
  BEGIN
    IF RESET THEN
      INIT;
  END;
```

```
(*****
(*)
(*) 5. INTERRUPT FROM CONTROLLER ROUTINE (*)
(*****)
```

PROCEDURE INTERRU;

```
  BEGIN
  END;
```

00/1.2 AAEC (1ST FEB 78)

```
(*****
(*)
(*) 6. SVC SUPERVISOR. ALL ROUTINE (*)
(*****
(*)
(*) START A TASK ROUTINE (SATASK) (*)
(*) 1. SET THE RUN FLAG IN TCB IF HALT FLAG IS NOT SET. (*)
(*) 2. SET SCH FLAG, SO THE SCHEDULER IS EXECUTED. (*)
(*) 3. THE USER'S ZERO FLAG IS SET TRUE IF TASK IS SET TO (*)
(*) RUN. FALSE IF THE TASK WAS HALTED AND CAN NOT BE RUN. (*)
(*) 4. THE USER'S ZERO FLAG IS SET FALSE IF THE TASK ID IS (*)
(*) NOT FOUND (*)
(*****)
```

PROCEDURE SATASK;

NAVTRAEQUIPCEN 78-C-0157-1

```

VAR
  ID:INTEGER;
  FLAG1:BOOLEAN;
BEGIN
  ID:=USERSP0.PC0.NEXT0.BYTE;
  USERSP0.PC:=USERSP0.PC0.NEXT0.NEXT;
  FLAG1:=FALSE;
  TEMPJQ:=STARTJQ;
  REPEAT
    IF TEMPJQ0.ID=ID THEN
      BEGIN
        FLAG1:=TRUE;
        IF :TEMPJQ0.TCB0.HALT THEN
          BEGIN
            TEMPJQ0.TCB0.RUN:=-TRUE;
            SCH:=-TRUE;
            USERSP0.CC.ZERO:=TRUE;
          END
        ELSE USERSP0.CC.ZERO:=-FALSE;
      END
    ELSE
      BEGIN
        TEMPJQ:=TEMPJQ0.NEXT;
        IF TEMPJQ0.NEXT=NIL THEN
          BEGIN
            FLAG1:=TRUE;
            USERSP0.CC.ZERO:=-FALSE;
          END;
        END;
      END
    UNTIL FLAG1;
  END;

```

```

(*****
(* HOLD A TASK ROUTINE (HATASK) *)
(* 1. SET THE HOLD FLAG IN THE TCB FOR A TASK. *)
(* 2. IF THE TASK ID IS VALID, THE SCH FLAG IS SET AND THE *)
(* USERS ZERO FLAG IS SET. *)
(* 3. IF THE TASK ID IS NOT VALID, THE USERS ZERO FLAG IS *)
(* FALSE. *)
(*****

```

00/1.2 AAEC (1ST FEB 78)

```

PROCEDURE HATASK;
VAR
  ID:INTEGER;
  FLAG1:BOOLEAN;
BEGIN
  ID:=USERSP0.PC0.NEXT0.BYTE;
  USERSP0.PC:=USERSP0.PC0.NEXT0.NEXT;
  FLAG1:=FALSE;
  TEMPJQ:=STARTJQ;
  REPEAT
    IF TEMPJQ0.ID=ID THEN
      BEGIN

```

```

FLAG1:=TRUE;
TEMP100:=TEMP100.NEXT;
SCH:=TRUE;
USERSP0.CO.ZERO:=TRUE;
END
ELSE
BEGIN
TEMP100:=TEMP100.NEXT;
IF TEMP100.NEXT=000 THEN
BEGIN
FLAG1:=TRUE;
USERSP0.CO.ZERO:=FALSE;
END
END;
UNTIL FLAG1;
END;

```

```

(*****
(* READ A FLAG ROUTINE (READFLAG) *)
(* 1. THIS ROUTINE READS ONE OF EIGHT FLAGS. IF THE FLAG IS *)
(* SET THE USERS ZERO FLAG IS SET TRUE. IF THE FLAG IS *)
(* NOT SET THE USERS ZERO FLAG IS SET FALSE. *)
(* 2. THE USER P.C. IS INCREMENTED TO THE NEXT INSTRUCTION. *)
(*****

```

PROCEDURE READFLAG;

```

VAR
ID:INTEGER;
BEGIN
ID:=USERSP0.POP.NEXT0.BYTE;
USERSP0.PC:=USERSP0.POP.NEXT0.NEXT;
IF USERFLAGID THEN
USERSP0.CO.ZERO:=TRUE
ELSE
USERSP0.CO.ZERO:=FALSE;
END;

```

```

(*****
(* SET A FLAG (SETFLAG) *)
(* 1. THIS ROUTINE SETS ONE OF EIGHT FLAGS TO TRUE. *)
(* 2. IF A FLAG GREATER THAN 8 IS GIVEN THE USERS ZERO FLAG *)
(* IS SET TO FALSE. *)
(*****

```

PROCEDURE SETFLAG;

```

VAR
00/1.2  ASEP (1ST FEB 78)

ID:INTEGER;
BEGIN
ID:=USERSP0.POP.NEXT0.BYTE;
USERSP0.PC:=USERSP0.POP.NEXT0.NEXT;
IF ID>8 THEN

```

```

BEGIN
  USERFLAGCIDI:=TRUE;
  USERSP0.CC.ZERO:=TRUE;
END
ELSE
  USERSP0.CC.ZERO:=FALSE;
END;

```

```

(*****
(* RESET A FLAG (RESFLAG) *)
(* 1. THIS ROUTINE RESET OR SET ONE OF EIGHT FLAGS TO FALSE. *)
(* 2. IF A FLAG GREATER THAN 0 IS GIVEN THE USERS ZERO FLAG *)
(* IS SET TO FALSE. *)
(*****

```

```

PROCEDURE RESFLAG;
VAR
  ID:INTEGER;
BEGIN
  ID:=USERSP0.PC0.NEXT0.BYTE;
  USERSP0.PC:=USERSP0.PC0.NEXT0.NEXT;
  IF ID<=8 THEN
    BEGIN
      USERFLAGCIDI:=FALSE;
      USERSP0.CC.ZERO:=TRUE;
    END
  ELSE USERSP0.CC.ZERO:=FALSE;
END;

```

```

(*****
(* READ A LOCATION (READLOC) *)
(* 1. READ ANY LOCATION IN MEMORY AND PLACE CONTENTS IN THE *)
(* USERS REG. A. *)
(* 2. USERS ZERO FLAG IS SET TO TRUE. *)
(*****

```

```

PROCEDURE READLOC;
BEGIN
  USERSP0.REGA:=USERSP0.PC0.NEXT0.POINTER0.BYTE;
  USERSP0.CC.ZERO:=TRUE;
  USERSP0.PC:=USERSP0.PC0.NEXT0.NEXT0.NEXT;
END;

```

```

(*****
(* START A TASK ON CONDITION (STASKC) *)
(* 1. THIS ROUTINE CHECKS ONE OF EIGHT FLAGS FOR A TRUE *)
(* CONDITION BEFORE STARTING THE TASK. IF THE FLAG IS TRUE *)
(* THE JOB IS FLAGED TO RUN AND THE USERS ZERO FLAG IS SET *)
(* TRUE. IF THE FLAG IS FALSE, THE STATE OF THE TASK IS *)
(* NOT CHANGED AND THE USERS ZERO FLAG IS SET FALSE. *)
(*****

```

00/1.2 AAEC (1ST FEB 78)

PROCEDURE STASKC;

```

VAR
  ID:INTEGER;
BEGIN
  ID:=USERSPP.PC@.NEXT@.BYTE;
  IF ID=8 THEN
    BEGIN
      IF USERFLAGID THEN
        BEGIN
          USERSPP.PC:=USERSPP.PC@.NEXT@.NEXT;
          SATASK;
        END
      ELSE
        BEGIN
          USERSPP.PC.ZERO:=FALSE;
          USERSPP.PC:=USERSPP.PC@.NEXT@.NEXT;
        END
      END
    END
  ELSE
    BEGIN
      USERSPP.PC.ZERO:=FALSE;
      USERSPP.PC:=USERSPP.PC@.NEXT@.NEXT;
    END
  END;
END;

```

```

(*****
(* HOLD A TASK ON CONDITION (HTASKC) *)
(* 1. THIS ROUTINE CHECKS ONE OF EIGHT FLAGS FOR A TRUE *)
(* CONDITION BEFORE HOLDING THE TASK. IF THE FLAG IS TRUE *)
(* THE JOB IS FLAGGED TO HOLD AND THE USERS ZERO FLAG IS SET *)
(* TRUE. IF THE FLAG IS FALSE, THE STATE OF THE TASK IS NOT *)
(* CHANGED AND THE USERS ZERO FLAG IS SET FALSE. *)
(*****

```

PROCEDURE HTASKC;

```

VAR
  ID:INTEGER;
BEGIN
  ID:=USERSPP.PC@.NEXT@.BYTE;
  IF ID=8 THEN
    BEGIN
      IF USERFLAGID THEN
        BEGIN
          USERSPP.PC:=USERSPP.PC@.NEXT@;
          HATASK;
        END
      ELSE
        BEGIN
          USERSPP.PC.ZERO:=FALSE;
          USERSPP.PC:=USERSPP.PC@.NEXT@;
        END
      END
    END
  END;
END;

```



```

      END
    ELSE
      BEGIN
        USERSP@.CC.ZERO:=FALSE;
        USERSP@.PC:=USERSP@.PC@.NEXT@.NEXT;
      END
    END;
  END;
00/1.2   AAEC (1ST FEB 78)

```

```

(*****
(*  START A TASK AT A CERTAIN TIME (STASKT)
(*  1. THIS ROUTINE PLACES AN OPCODE ON THE EVENT QUEUE, THAT
(*  WILL START A TASK AT A CERTAIN TIME. MEMBERS IN THE
(*  QUEUE ARE PLACED IN CHRONOLOGICAL ORDER, THEREFORE WHEN
(*  THE EVENT TIMER CAUSES AN INTERRUPT, THE FIRST ENTRY WILL
(*  ALWAYS BE SERVICED.
(*  2. ADDRESSES OF EMPTY SPACES IN THE QUEUE ARE KEPT IN AN
(*  ARRAY TO MINIMIZE SEARCHING FOR FREE SPACE.
(*  3. THE MAXIMUM SIZE OF THE QUEUE IS SET BY EVMAX.
(*****

```

```

PROCEDURE STASKT;
VAR
  TT, ID: INTEGER;
  FLAG1: BOOLEAN;
BEGIN
  ID:=USERSP@.PC@.NEXT@.BYTE;
  TT:=USERSP@.PC@.NEXT@.NEXT@.BYTE;
  USERSP@.PC:=USERSP@.PC@.NEXT@.NEXT@.NEXT;
  IF TT>ABSTIME THEN
    BEGIN
      IF EEMPTY THEN
        BEGIN
          FIRSTEQ:=FREE(CUR);
          FIRSTEQ@.NEXT:=NIL;
          CUR:=CUR-1;
          EEMPTY:=FALSE;
          FIRSTEQ@.ID:=ID;
          FIRSTEQ@.TIME:=TT;
          FIRSTEQ@.OPCODE:=0;
          TIMERE:=TT-ABSTIME;
        END
      ELSE
        BEGIN
          FLAG1:=FALSE;
          TEMPEQ:=FIRSTEQ;
          TEMPEQ2:=NIL;
          REPEAT
            IF TEMPEQ@.TIME>TT THEN

```

AD-A080 735

SOUTH CAROLINA UNIV COLUMBIA DEPT OF ELECTRICAL AND --ETC F/6 9/2
MULTIPLE MICROCOMPUTER CONTROL ALGORITHM.(U)
SEP 79 R D PETTUS, R D BONNELL, M N HUMNS N61338-78-C-0157

UNCLASSIFIED

NAVTRAEQUIPC-78-C-0157-1 NL

3 OF 3

67
2080715



END
DATE
FILED
3-80
DOC

```

      BEGIN
      FLAG1:=TRUE;
      IF TEMPEQ2 = NIL THEN
      BEGIN
      FREE(CUR10.NEXT:=FIRSTEQ;
      FIRSTEQ:=FREE(CUR1;
      TEMPEQ2:=FIRSTEQ;
      TIMERE:=TT-ABSTIME;
      END
      ELSE
      BEGIN
      FREE(CUR10.NEXT:=TEMPEQ0.NEXT;
      TEMPEQ20.NEXT:=FREE(CUR1;
      END;
      CUR:=CUR-1;
      TEMPEQ20.ID:=ID;
      TEMPEQ20.TIME:=TT;
00/1.2      AAEC (1ST FEB 78)

      TEMPEQ20.OPCODE:=0;
      END
      ELSE
      BEGIN
      IF TEMPEQ0.NEXT=NIL THEN
      BEGIN
      TEMPEQ0.NEXT:=FREE(CUR1;
      FREE(CUR10.NEXT:=NIL;
      CUR:=CUR-1;
      TEMPEQ0.NEXT0.ID:=ID;
      TEMPEQ0.NEXT0.TIME:=TT;
      TEMPEQ0.NEXT0.OPCODE:=0;
      FLAG1:=TRUE;
      END
      ELSE
      BEGIN
      TEMPEQ2:=TEMPEQ;
      TEMPEQ:=TEMPEQ0.NEXT;
      END;
      END
      UNTIL FLAG1;
      USERSP0.CC.ZERO:=TRUE;
      END
      ELSE
      USERSP0.CC.ZERO:=FALSE;
      END;

      (*****
      (* START A TASK AFTER AN INTERVAL (TASKIN) *)
      (* 1. THIS ROUTINE TAKES AN INTERVAL OF TIME AND CONVERTS IT *)
      (* TO ABSOLUTE TIME, THEN CALLS STASKT WHICH THEN PLACES *)
      (* THE EVENT ON THE QUEUE. *)
      (*****)

```

NAVTRAEQUIPCEN 78-C-0157-1

```

PROCEDURE STASKIN;
  VAR TT:INTEGER;
  BEGIN
    TT:=USERSP0.PC0.NEXT0.NEXT0.BYTE;
    TT:=TT + ABSTIME;
    USERSP0.PC0.NEXT0.NEXT0.BYTE:=TT;
    STASK;
  END;

```

```

(*****
(*) HALT A TASK FOREVER (HALTASK) (*)
(*) 1. THIS ROUTINE SETS THE HALT FLAG FOR TASK. THIS HALTS THE (*)
(*) TASK FOREVER OR UNTIL THE CONTROL PROCESSOR RESTARTS IT. (*)
(*) 2. THE LOCAL ATM CAN NOT RESTART IT. (*)
(*****)

```

```

PROCEDURE HALTASK;
  VAR
    ID :INTEGER;
    FLAG1:BOOLEAN;
  BEGIN
    ID:=USERSP0.PC0.NEXT0.BYTE;
    USERSP0.PC:=USERSP0.PC0.NEXT0.NEXT;
    00/1.2 AAEC (1ST FEB 78)

    FLAG1:=FALSE;
    TEMPJQ:=STARTJQ;
    REPEAT
      IF TEMPJQ0.ID=ID THEN
        BEGIN
          FLAG1:=TRUE;
          TEMPJQ0.TCRP0.HALT:=TRUE;
          SCH:=TRUE;
          USERSP0.CC.ZERO:=TRUE;
        END
      ELSE
        BEGIN
          TEMPJQ:=TEMPJQ0.NEXT;
          IF TEMPJQ=NIL THEN
            BEGIN
              FLAG1:=TRUE;
              USERSP0.CC.ZERO:=FALSE;
            END
          END;
        UNTIL FLAG1;
      END;

```

```

(*****
(*) READ THE PRIORITY OF A TASK (READPRI) (*)
(*) 1. THIS ROUTINE READS THE PRIORITY OF A TASK AND RETURNS IT (*)
(*) TO THE USER IN REGISTER A. (*)
(*****)

```

PROCEDURE READPRI:

```

VAR
  ID : INTEGER;
  FLAG1: BOOLEAN;
BEGIN
  ID:=USERSP0.PC0.NEXT0.BYTE;
  USERSP0.PC1:=USERSP0.PC0.NEXT0.NEXT0;
  FLAG1:=FALSE;
  TEMPJQ:=STARTJQ;
  REPEAT
    IF TEMPJQ.ID = ID THEN
      BEGIN
        FLAG1:=TRUE;
        USERSP0.CC.ZERO:=TRUE;
        USERSP0.REGAL:=TEMPJQ.TCRP0.PRI0;
      END
    ELSE
      BEGIN
        TEMPJQ:=TEMPJQ.NEXT;
        IF TEMPJQ = NIL THEN
          BEGIN
            FLAG1:=TRUE;
            USERSP0.CC.ZERO:=FALSE;
          END;
        END;
      END
    UNTIL FLAG1;
  END;

```

```

(*****
(*) ASSIGN A PRIORITY TO A TASK (ASSPRI0)
00/1.2      AAFU (1ST FEB 78)

```

```

(*) 1. THIS ROUTINE WILL CHANGE THE PRIORITY OF A TASK TO ANY (*)
(*) PRIORITY BETWEEN 0 AND 255. IF A PRIORITY DOESNT EXIST (*)
(*) ONE WILL BE CREATED. (*)
(*) 2. TASKS WILL BE ENTERED AHEAD OF ALL OTHER TASK THAT HAVE (*)
(*) THE SAME PRIORITY. (*)
(*) 3. IF THE PRIO OF A TASK REMAINS THE SAME, THE TASK WILL (*)
(*) STILL BE MOVED TO THE HEAD OF ITS PRIORITY. (*)
(*****

```

PROCEDURE ASSPRI0:

```

VAR
  ID,PRI : INTEGER;
  FLAG1 : BOOLEAN;
BEGIN
  ID:=USERSP0.PC0.NEXT0.BYTE;
  PRI:=USERSP0.PC0.NEXT0.NEXT0.BYTE;
  USERSP0.PC1:=USERSP0.PC0.NEXT0.NEXT0.NEXT0;
  FLAG1:=FALSE;
  TEMPJQ:=STARTJQ;

```

NAVTRAEQUIPCEN 78-C-0157-1

```

REPEAT
  IF TEMPJQ0.ID=ID THEN
    BEGIN
      TEMPPQ2:=TEMPJQ0.PRIOR;
      TEMPPQ:=TEMPJQ0.FDP;
      TEMPPQ20.BKP0.FDP:=TEMPPQ20.FDP;
      TEMPPQ20.FDP0.BKP:=TEMPPQ20.BKP;
      TEMPJQ0.PRI0:=PRI;
      TEMPPQ20.PRI0:=PRI;
      TEMPPQ20.TCBP0.PRI0:=PRI;
      IF PRI=0 THEN
        BEGIN
          TEMPPQ20.BKP:=NIL;
          TEMPPQ20.FDP:=STARTPQ;
          STARTPQ0.BKP:=TEMPPQ;
          STARTPQ:=TEMPPQ2;
        END
      ELSE
        BEGIN
          FLAG1:=FALSE;
          IF PRI>TEMPPQ0.PRI0 THEN
            TEMPPQ:=ENDPQ;
          REPEAT
            IF TEMPPQ0.PRI0<PRI THEN
              BEGIN
                FLAG1:=TRUE;
                TEMPPQ20.BKP:=TEMPPQ;
                TEMPPQ20.FDP:=TEMPPQ0.FDP;
                TEMPPQ0.FDP:=TEMPPQ2;
                IF TEMPPQ20.FDP=NIL THEN
                  ENDPQ:=TEMPPQ2;
                ELSE
                  TEMPPQ20.FDP0.BKP:=TEMPPQ2;
                END
              END
            ELSE
              TEMPPQ:=TEMPPQ0.BKP;
            UNTIL FLAG1;
          END
        END
      END
    END
  00/1.2  AAEC (1ST FEB 78)
  ELSE
    BEGIN
      TEMPJQ:=TEMPJQ0.NEXT;
      IF TEMPJQ=NIL THEN
        BEGIN
          FLAG1:=TRUE;
          USERSP0.CC.ZERO:=FALSE;
        END
      END
    UNTIL FLAG1;
  END;

```

```

(*****
(*  SET WRITE FLAG FOR A SHARED VAR (SETWRIT)      *)
(*****

```

```

PROCEDURE SETWRIT;
BEGIN
END;

```

```

(*****
(*  STATUS OF A TASK (STATUS)                      *)
(*****

```

```

PROCEDURE STATUS;
BEGIN
END;

```

```

(*****
(*  TIME OF DAY (TOD)                              *)
(*****

```

```

PROCEDURE TOD;
BEGIN
END;

```

```

PROCEDURE SVC;

```

```

VAR

```

```

  ID      : INTEGER;
  FLAG1   : BOOLEAN;

```

```

BEGIN

```

```

  CASE USERSPP.PCB.BYTE OF

```

```

    0 : SATASK :      (*****
                      (* START A TASK                      *)
    1 : HALASK :      (* HOLD A TASK                       *)
    2 : READFLG :      (* READ  FLAGS                      *)
    3 : SETFLG :      (* SET   FLAGS                      *)
    4 : RESETFLG :    (* RESET  FLAGS                     *)
    5 : READLOC :      (* READ A LOCATION IN MEMORY        *)
    6 : STASKC :      (* START A TASK CONDITIONALLY        *)
    7 : HTASKC :      (* HOLD A TASK CONDITIONALLY         *)
    8 : STASKT :      (* START A TASK AT A CERTAIN TIME    *)
    9 : STASKIN :     (* START A TASK AFTER AN INTERVAL    *)
   10 : HALTASK :     (* STOP A TASK FOREVER               *)
   11 : READPRI :     (* READ THE PRIORITY OF A TASK       *)

```

```

00/1.2  AAEC (1ST FEB 78)

```

```

    12 : ASSPRIOR :   (* ASSIGN A PRIORITY TO A TASK      *)
    13 : SETWRIT :   (* SET WRITE FLAG FOR A SHARED VAR  *)
    14 : STATUS :    (* STATUS OF A TASK                  *)
    15 : TOD :       (* TIME OF DAY                       *)
  END;
(*****

```

END;

```

(*****
(*****
(*****
(*****  MAIN PROGRAM
(*****
(*****
(*****
(*****
(*****
(*****

```

```

(*****
(*)
(*)  INTERRUPT HANDLER AND VECTOR TABLE
(*)
(*)
(*****

```

BEGIN

INTER:=10;

CASE INTER OF

10 : AHALT;

11 : BEGIN;

RESETF:=TRUE;

COMM;

END;

20 : COMM ;

30 : BEGIN

OSENTRY;

FLAG:=FALSE;

TEMPSYQ:=STARTSYQ;

REPEAT

IF TEMPSYQ.ID=2 THEN

BEGIN

TEMPSYQ.RUN:=TRUE;

SYSF:=TRUE;

FLAG:=TRUE;

END

ELSE TEMPSYQ:=TEMPSYQ.NEXT;

UNTIL FLAG;

END;

40 : BEGIN

OSENTRY;

STARTSYQ.RUN:=TRUE;

END;

41 : BEGIN

OSENTRY;

EVENTF:=TRUE;

END;

NAVTRAEQUIPCEN 78-C-0157-1

```

42 : BEGIN
    OSENTRY;                                     (*****
00/1.2  AAEC (1ST FEB 78)                       (*****

    TEMPSYQ:=STARTSYQ;                          (*          *)
    FLAG:=FALSE;                                (* THIS SYSTEM JOB STOPS A *)
    REPEAT                                       (* USER JOB IF IT RUNS OUT *)
        IF TEMPSYQ.ID=3 THEN                   (* OF TIME.                *)
            BEGIN                               (*          *)
                TEMPSYQ.RUN:=TRUE;             (*****
                SYSF:=TRUE;
                FLAG:=TRUE;
            END
        ELSE TEMPSYQ:=TEMPSYQ.NEXT;
    UNTIL FLAG;
END;
50 : BEGIN
    OSENTRY;
    INTERR;
END;
60 : BEGIN
    OSENTRY;
    SVC ;
END;
END;

(*****
(**          **)
(** EVENT ROUTINE.          **)
(**          **)
(*****

IF EVENT THEN
BEGIN
    EVENT:=FALSE;
    CUR:=CUR+1;
    FIRSTEQ:=FIRSTEQ;
    TEMPEQ:=FIRSTEQ;
    IF FIRSTEQ.NEXT=NIL THEN
        EMPTY:=TRUE
    ELSE
        FIRSTEQ:=FIRSTEQ.NEXT;
    TIME:=TEMPEQ.TIME+ABSTIME;
    CASE TEMPEQ.OPCODE OF

        0 : BEGIN                                (*****
            TEMPJOB:=STARTJOB;                  (*          *)
            FLAG:=FALSE;                        (* START A JOB*)
            REPEAT                               (* OPERATION *)
                IF TEMPEQ.ID=TEMPJOB.ID THEN    (*          *)
                    BEGIN                       (*****
                        FLAG:=TRUE;
                        IF TEMPEQ.TCOPR.HALT THEN
                            BEGIN
                                TEMPJOB.TCOPR.RUN:=TRUE;
                                TEMPJOB.TCOPR.HOLD:=FALSE;
                                SCH:=TRUE;
                            END;
                    END;
            END;

```

NAVTRAEQUIPCEN 78-C-0157-1

```

                                END
                                ELSE TEMPJQ:=TEMPJQ.NEXT;
                                UNTIL FLAG=TRUE;
                                END;
00/1.2    AAEC (1ST FER 78)

                                END;
                                END; (* END CASE. MORE CAN BE ADDED *)

(*****
(*)
(*)    EXECUTE SYSTEM JOBS IF ANY ARE READY
(*)
(*)
(*****)

IF SYSF THEN
BEGIN
    SYSF:=FALSE;
    TEMPSYQ:=STARTSYQ;
    REPEAT
        IF TEMPSYQ.RUN THEN
        BEGIN
            TEMPSYQ.TCBP.ACTIVE:=TRUE;
            WITH TEMPSYQ.TCBP DO
            BEGIN
                STACKA.X:=ORGST.X;
                STACKA.Y:=ORGST.Y;
                STACKA.USP:=ORGST.USP;
                STACKA.PC:=ORGST.PC;
            END;
            ATMSP:=SP;
            SP:=TEMPSYQ.TCBP.STACKA;
            RTI;
            TEMPSYQ.TCBP.STACKA:=SP;
            SP:=ATMSP;
            TEMPSYQ.TCBP.ACTIVE:=FALSE;
        END;
        TEMPSYQ:=TEMPSYQ.NEXT;
    UNTIL TEMPSYQ=NIL;
END;

(*****
(*)
(*)    TASK SCHEDULER ROUTINE
(*)
(*)
(*****)

```

NAVTRAFQUIPCEN 78-C-0157-1

```

IF SCH THEN
  BEGIN
    SCH:=FALSE;
    IF ACTJOB.TCBP=NIL THEN USAVED:=TRUE
    ELSE
      BEGIN
        IF ACTJOB.TCBP.HOLD OR ACTJOB.TCBP.HALT THEN
          BEGIN
            ACTJOB.TCBP.ACTIVE:=FALSE;
            ACTJOB.TCBP.RUN:=FALSE;
            USAVED:=TRUE;
          END
        ELSE USAVED:=FALSE;
      END;
    IF USAVED THEN
      TEMPPQ:=CURPQ
    ELSE

```

00/1.2

AAEC (1ST FEB 78)

```

      TEMPPQ:=STARTPQ;
      FLAG:=FALSE;
      REPEAT
        IF TEMPPQ.TCBP.RUN THEN FLAG:=TRUE
        ELSE TEMPPQ:=TEMPPQ.NEXT;
      UNTIL FLAG;
      IF TEMPPQ.TCBP.ID=ACTJOB.ID THEN
        BEGIN
          CURPQ:=TEMPPQ;
          IF USAVED THEN
            BEGIN
              ACTJOB.TCBP.INTER:=TRUE;
              ACTJOB.TCBP.TIMER:=TIMER;
              ACTJOB.TCBP.STACK:=USERSP;
            END;
          IF TEMPPQ.TCBP.INTER THEN
            BEGIN
              ACTJOB.ID:=TEMPPQ.TCBP.ID;
              ACTJOB.TCBP:=TEMPPQ.TCBP;
              IF TEMPPQ.TCBP.UTIME=0 THEN
                TIMER:=99999
              ELSE
                TIMER:=TEMPPQ.TCBP.TIMER;
              USERSP:=TEMPPQ.TCBP.STACK;
              TEMPPQ.TCBP.ACTIVE:=TRUE;
              TEMPPQ.TCBP.INTER:=FALSE;
            END
          ELSE

```

NAVTRAEQUIPCEN 78-C-0157-1

```
BEGIN
  ACTJOB.ID:=TEMPPQB.TCBPQ.ID;
  ACTJOB.TCBP:=TEMPPQB.TCBP;
  IF TEMPPQB.TCBPQ.TOTTIME = 0 THEN
    TIMERJ:=99999
  ELSE
    TIMERJ:=TEMPPQB.TCBPQ.TOTTIME;
    TEMPPQB.TCBPQ.ACTIVE:=TRUE;
    WITH TEMPPQB.TCBPQ DO
      BEGIN
        STACKAQ.PC:=ORGSTQ.PC;
        STACKAQ.USP:=ORGSTQ.USP;
      END;
    USERSP:=TEMPPQB.TCBPQ.STACKAQ;
  END;
END;
END;
```

END.

L COMPILATION CONCLUDED *

DETECTED IN PASCAL PROGRAM *

DISTRIBUTION LIST

| | | |
|---|------|--|
| Naval Training Equipment Center Orlando, Florida 32813 | (35) | Naval Air Systems Command AIR 413 Washington, DC 20350 |
| Defense Documentation Center Cameron Station Alexandria, VA 22314 | (12) | AF ASD/ENE ATTN: Mr. A. Doty Wright-Patterson AFB, OH 45433 |
| University of South Carolina College of Engineering Electrical and Computer Engineering Columbia, SC 29208 | (8) | AF HRL/ASM ATTN: Mr. Don Gum Wright-Patterson AFB, OH 45433 |
| ALL OTHERS RECEIVE ONE COPY | | AF ASD/ENO ATTN: Mr. Phil Babel Wright-Patterson AFB, OH 45433 |
| Chief of Naval Education & Training Code N-331 Pensacola, FL 32508 | | |
| Naval Air Systems Command AIR 340D Washington, DC 20350 | | |
| Naval Air Systems Command AIR 340F Washington, DC 20350 | | |
| Naval Air Systems Command AIR 360B Washington, DC 20350 | | |
| Naval Air Systems Command Library AIR 50174 Washington, DC 20350 | | |
| Chief of Naval Material, Navy Dept Code MAT 08T Washington, DC 20360 | | |
| Chief of Naval Material, Navy Dept MAT 08Y Washington, DC 20360 | | |
| Chief of Naval Material, Navy Dept MAT 09Y Washington, DC 20360 | | |
| Chief of Naval Research ONR 461 800 North Quincy St Arlington, VA 22217 | | |